

KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

TECHNIQUES FOR REASONING IN FO(ID) AND FIXPOINT LOGIC

Ping Hou

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering

March 2010

TECHNIQUES FOR REASONING IN FO(ID) AND FIXPOINT LOGIC

Ping Hou

Jury:
Prof. Dr. Jean Berlamont, president
Prof. Dr. Marc Denecker, promotor
Prof. Dr. Maurice Bruynooghe
Prof. Dr. Walter Van Assche
Prof. Dr. Eugenia Ternovska (Simon Fraser University, Canada)
Prof. Dr. Hans Tompits (Vienna University of Technology, Austria)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

U.D.C. 681.3*I2

March 2010

© Katholieke Universiteit Leuven – Faculty of Engineering
Arenbergkasteel, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2010/7515/29
ISBN 978-94-6018-190-0

Abstract

Knowledge representation and reasoning is an area in Artificial Intelligence that is concerned with how to use a formal language to represent a domain of discourse. The key problem in knowledge representation is to develop formal languages (logics) that can be used to express a wide range of problems, to develop inference methods for these languages, and to develop efficient implementations of these inference methods.

FO(ID), also called ID-Logic, is a knowledge representation language that extends classical logic with inductive definitions. A large variety of practical problems can be expressed in FO(ID) in an intuitive way, and therefore FO(ID) has been promoted as a useful knowledge representation language.

In the first part of this thesis we study formal proof systems for FO(ID). The systems are formulated as sequent calculi for a propositional fragment of FO(ID), called PC(ID), and for FO(ID), respectively. The default approach to reasoning with inductive definitions is to formulate the induction principles of the inductively defined relations as suitable inference rules or axioms, which are incorporated into the reasoning framework of choice. Our proof systems adopt this direct approach with the definition rules formulated as rules for introducing atomic formulas involving defined predicates and inductive definitions on both the left and the right of sequents. In our proof system for FO(ID), we adopt the fixpoint expressions to represent unfounded sets, leading to the logic FO(ID,SLFP), which is an integration of least fixpoint expressions and FO(ID). We show this proof system to be sound with respect to a slightly restricted fragment of FO(ID) and to be complete for a more restricted fragment of FO(ID).

In the second part of this thesis we extend FO with fixpoint definitions. Our motivation for this is that, first, from a practical point of view, fixpoint definitions extend the expressivity of FO and allow to express a number of useful and practical concepts that cannot (or only very unpleasantly) expressed using FO(ID); second, from a theoretical point of view, we want to study the relationship between non-monotone induction in FO(ID) and the constructive principle underlying fixpoint definitions. The fixpoint definition construct in our new logic FO(FD) is a syntactic

variant of standard fixpoint logic and differs from it in two (minor) ways: an LP-like rule notation is used for expressing inductive and coinductive definitions, and the definitions define predicate constants rather than predicate variables. This modification facilitates the comparison and integration with FO(ID) and often improves knowledge representation.

We define FO(FD) and present some partial results on its relationship with FO(ID) and with Coinductive logic programming. The main work of this part is concerned with the propositional fragment of FO(FD), called PC(FD), and with finite model generation algorithms for it. To this aim, we provide an alternative semantical characterization of PC(FD), which yields important insights into the underlying structure of PC(FD) theories, and which therefore contributes to the understanding of the model generation task for PC(FD). We then study practical model generation algorithms for PC(FD). We also implement a prototype model generator for PC(FD).

Acknowledgements

I would like to express my gratitude to all people who helped me with the realization of this thesis.

First and foremost, most of my gratitude goes to Prof. Marc Denecker for being a great supervisor. I am grateful for the opportunity he gave me to start a Ph.D. I very much appreciate his knowledge and enthusiasm about my research topics. Whenever the results were less encouraging, a discussion with Marc gave me a new energy boost to go on. His influence on the contents of this thesis is omnipresent. His persistence on clarity and precision makes this thesis a lot more read-worthy. It is a great pleasure working with such a stimulating and enthusiastic mentor.

I am grateful to prof. Maurice Bruynooghe for proofreading this text and suggesting many improvements. I am still amazed about the speed with which he is able to actually suggest useful improvements in new scientific material.

I would also like to thank the other members of my jury, prof. Eugenia Ternovska, prof. Hans Tompits, prof. Walter Van Assche, and the chairman prof. Jean Berlamont, for taking time out of their busy schedule to read this work and attend my defense. Their detailed comments on the draft turned out to be important for improving the text.

A dissertation is usually not made by only one person. Therefore, I would like to thank my co-authors without whom some parts of this text would not have been possible: Johan Wittocx and Broes De Cat.

Thanks go to some other (ex-)members of the KR&R group not only for helpful academic discussions, but also for pleasant coffee breaks together: Joost Vennekens, Álvaro Cortés, Maarten Mariën, Stephen Bond and Hanne Vlaeminck.

I would like to express my deep gratitude to my best friend Yifei and her husband Feng. I can not imagine the life without their care, continued support and love. They and their adorable baby daughter Emma are my friends, my family, my insides.

I would like to sincerely thank all other friends for their support. By writing a long list I run the very real risk of accidentally leaving someone out, so I will simply say that you all know who you are: thank you.

I saved the last "thank you" for my mom and dad, the most important persons in my life. Even from 8789.4km away I've felt, as if they were all right here, the patience, the never-ending support, the full attention and the love of my parents. Without them, I probably would not have started a Ph.D. in the first place. To my mom, thanks for being there for me all these years.

Ping Hou
Leuven, March 2010

Contents

Abstract i

Acknowledgements iii

Contents v

List of Symbols ix

1 Introduction 1

1.1 General background and goal of this thesis 1

1.2 Contributions 6

1.3 Structure of the text 8

2 Preliminaries 11

2.1 Introduction 11

2.2 Classical Logic Formalisms 11

2.2.1 Propositional Logic 12

2.2.2 First-order Logic 14

2.2.3 Three-valued and Four-valued Semantics 19

2.3 Logic Programming 23

2.3.1 Well-founded Semantics 26

2.3.2 Stable model Semantics 27

2.4 FO(ID) 29
2.5 Sequent Calculus 32

3 LPC(ID): a proof system for PC(ID) 37
3.1 Introduction 37
3.2 Preliminaries 38
3.3 The deductive system for PC(ID) 39
3.4 Soundness of **LPC(ID)** 44
3.5 Completeness of **LPC(ID)** 52
3.6 Cut-elimination 62
3.7 Complexity results 64
3.8 Conclusions and related work 65

4 LFO(ID,SLFP): a proof system for FO(ID) based on least fixpoint logic 67
4.1 Introduction 67
4.2 Preliminaries 69
4.2.1 Normal form 69
4.2.2 Syntax of FO(ID,SLFP) 70
4.2.3 Semantics of FO(ID,SLFP) 71
4.3 The deductive system for FO(ID,SLFP) 74
4.4 Soundness of **LFO(ID,SLFP)** 78
4.5 Completeness of **LFO(ID,SLFP)** 83
4.6 Conclusions and related work 103

5 Extending FO(.) with induction and coinduction 105
5.1 Introduction 105
5.2 Induction and coinduction 106
5.3 Fixpoints in computer science 108

5.4 Fixpoint logics and FO(ID) 110
5.5 Formal Preliminaries 114
5.5.1 Orders, lattices, operators and fixpoints 114
5.5.2 Lattice congruences 116
5.5.3 Structure lattices 116
5.6 A logic of fixpoint definitions 117
5.6.1 Syntax 117
5.6.2 Semantics 120
5.7 A comparison of FO(FD) and FO(ID) 124
5.8 Coinductive logic programming vs. FO(FD) 128
5.9 Conclusion 130

6 SAT(FD) 133

6.1 Introduction 133
6.2 Preliminaries 134
6.2.1 Completion 134
6.2.2 Normal form 135
6.2.3 Some graph concepts 137
6.3 A standard algorithm for the computation of fixpoint expressions . 138
6.3.1 Introduction 138
6.3.2 The propositional μ-calculus 139
6.3.3 The algorithm 142
6.3.4 A standard algorithm for the computation of fixpoint definitions 153
6.4 Justification semantics of fixpoint definitions 159
6.4.1 Justifications 159
6.4.2 Properties of justifications 163
6.4.3 Three-valued semantics 171

6.4.4 Related work 172
6.5 SAT(FD) theory 172
6.6 Algorithms 176
6.6.1 UnitPropDef 177
6.6.2 An improved algorithm for the computation of fixpoint definitions 177
6.6.3 The algorithm for finding local witnesses 183
6.6.4 The algorithm for finding witnesses 186
6.7 MX(FO(FD)) 189
6.7.1 Model expansion 189
6.7.2 The MX(FO(FD)) system 190
6.7.3 Applications of FO(FD) and MX(FO(FD)) 192
6.8 Conclusions and related work 193

7 Conclusion **195**
7.1 Summary of contributions 195
7.2 Future work 196

A Extended Clausal Normal Form (ECNF) **199**
A.1 Inductive definitions 200
A.2 Fixpoint definitions 201

Bibliography **203**

List of Symbols

Σ	A vocabulary, page 12
$\mathbf{t}$	The truth value "true", page 12
$\mathbf{f}$	The truth value "false", page 12
I	An interpretation, page 12
$\leq$	The truth order, page 13
$\models$	The satisfaction relation between an interpretation and a theory, or the entailment relation between two theories, page 13
$t[x/u]$	The term obtained by substituting u for all occurrences of x in t, page 15
$Var(t)$	The set of variables appearing in t, page 15
$\neg S$	(S is a set of literals) The set $\{\neg l \mid l \in S\}$, page 15
$\widehat{S}$	(S is a set of literals) The set $S \cup \neg S$, page 15
$\varphi[x/t]$	The formula obtained by substituting t for all occurrences of x in φ, page 15
$free(\varphi)$	The set of free variables occurring in φ, page 16
$dom(I)$	The domain of interpretation I, page 16
$P(\bar{a})$	A domain atom, page 16
$I[\bar{x}/\bar{a}]$	The interpretation that has the same domain as I, interprets $\bar{x}$ by $\bar{a}$ and coincides with I on all other symbols, page 17

$\mathcal{U}_\Sigma$	The Herbrand universe of Σ, page 18
$\mathcal{B}_\Sigma$	The Herbrand base of Σ, page 18
$\mathcal{I}$	A Herbrand interpretation, page 18
$\mathbf{u}$	The truth value "unknown", page 19
$\mathbf{i}$	The truth value "inconsistent", page 19
$\mathcal{V}_3, \mathcal{V}_4$	Three- or four-valued truth values, page 20
$\leq_p$	The precision order, page 20
$\mathcal{T}_P$	The immediate consequence operator on P, page 25
$I[P(\bar{a})/\mathbf{f}]$	The interpretation I' that is identical to I except that $I'(P(\bar{a})) = \mathbf{f}$. Similar for $\mathbf{t}$, page 26
$I[U/\mathbf{f}]$	The interpretation I' that is identical to I except that $I'(P(\bar{a})) = \mathbf{f}$ for each $P(\bar{a}) \in U$. Similar for $\mathbf{t}$, page 26
$\mathcal{GL}(P, I)$	Gelfond-Lifschitz reduct of P under I, page 28
$Def(D)$	The set of defined predicates of D, page 29
$Open(D)$	The set of open symbols of D, page 29
Γ, Δ	A set of formulas, page 33
S_i	A sequent, page 33
$\bot$	The logical constant "false", page 33
$\top$	The logical constant "true", page 33
T	A proof tree, page 35
φ_P	The body of the rule defining P, page 38
$\varphi^\triangleright$	The renaming of φ with respect to U, page 40
T_S	The reduction tree for S, page 52

Σ^c	A set of predicate constants and function symbols, page 70
Σ^v	A set of predicate variables, page 70
$\varphi(P/\rho)$	The formula obtained by substituting ρ for all free occurrences of P in φ, page 71
$[t]$	The equivalence class of t with respect to the relation $\sim$, page 90
$U_D(I)$	The set of all unfounded sets of D in I, page 94
$\mathcal{R}$	A set of rules, page 118
$\mathcal{D}$	A fixpoint definition, page 118
Δ	A least fixpoint definition, page 118
∇	A greatest fixpoint definition, page 118
$Open(\mathcal{D})$	The set of all defined predicates of $\mathcal{D}$, page 118
$Def(\mathcal{D})$	The set of all open symbols of $\mathcal{D}$, page 118
$\mathrm{lfm}_{\mathcal{D}}(I_O)$	Model of least fixpoint definition $\mathcal{D}$ extending $Open(\mathcal{D})$-interpretation I_O, page 121
$\mathrm{gfm}_{\mathcal{D}}(I_O)$	Model of greatest fixpoint definition $\mathcal{D}$ extending $Open(\mathcal{D})$-interpretation I_O, page 121
$M^{co}(P)$	The model of a coinductive logic program, page 129
$comp(\mathcal{D})$	Completion of definition $\mathcal{D}$, page 134
φ_l	The body of the rule defining l if l is an atom, or its negation if l is a negative literal. Also used to denote the set of literals in that body, page 135
$\mathbf{C}_{\mathrm{lits}}$	Conjunctively defined literals, page 135
$\mathbf{D}_{\mathrm{lits}}$	Disjunctively defined literals, page 135
$\bar{\imath}, \bar{\jmath}$	Vectors of iteration indices, page 145
$\mathcal{D}(a)$	The subdefinition of $\mathcal{D}$ in which a is locally defined, page 153
$DJ(l)$	Direct justification for literal l, page 159

J	A justification, page 160
$DJ_J(l)$	(J is a justification, l a literal) Direct justification for l in J, page 160
$d_J(l)$	($l \in \mathbf{D}_{\text{lits}}$, J is a justification) The literal x for which $DJ_J(l) = \{x\}$, page 160
$\mathbf{L}_{\text{lits}}$	Literals locally defined in a least fixpoint definition, page 161
$\mathbf{G}_{\text{lits}}$	Literals locally defined in a greatest fixpoint definition, page 161
$Sub(J, l)$	(J is a justification, l a literal) Subjustification of J starting in l, page 163
$\mathcal{L}$-MX	Model expansion for logic $\mathcal{L}$, page 189
$\mathcal{L}$-$MX_{\langle T,\sigma\rangle}$, $MX_{\langle T,\sigma\rangle}$	Parameterized model expansion, with theory T and input vocabulary σ (for logic $\mathcal{L}$), page 190

Chapter 1

Introduction

1.1 General background and goal of this thesis

This thesis falls within the broader research area of *knowledge representation and reasoning* (KR&R). In this domain, one studies knowledge: what types of knowledge there are, how they are used, how they can be expressed in a formal language, etc. An important goal of this field is to develop formal languages (logics) that can be used to express a wide range of problem domains, to develop automated reasoning methods for these languages, and to develop efficient implementations of these reasoning methods. This goal is an important step towards *declarative problem solving*: a method of (computational) problem solving whereby the human expert *declares* his knowledge of the problem in a formal specification (theory), and then simply applies a program that implements the appropriate automated reasoning task to this theory, to solve the problem.

Perhaps the two most important knowledge representation paradigms of the moment are on the one hand, classical logic-based approaches such as description logics [9], and on the other hand, rule-based approaches based on logic programming (LP) and extensions such as Answer Set Programming and Abductive Logic Programming [13, 94]. The latter disciplines are rooted firmly in the discipline of Non-Monotonic Reasoning [125]. FO(ID) [55], also called ID-Logic, integrates both paradigms in a tight, conceptually clean manner. The key to integrate "rules" into classical first-order logic (FO) is the observation that natural language, or more precisely, the informal language of mathematicians, has an informal rule-based construct: the construct of *inductive/recursive definitions* (IDs).

In Figure 1.1 and Figure 1.2, we displayed two prototypical examples of the

Definition 1.1.1 *The transitive closure T_G of a directed graph G is defined by induction:*

- $(x, y) \in T_G$ *if* $(x, y) \in G$;
- $(x, y) \in T_G$ *if for some vertex* z, $(x, z) \in T_G$ *and* $(z, y) \in T_G$.

Figure 1.1: Definition of Transitive closure

Definition 1.1.2 *The satisfaction relation $\models$ between σ-interpretations I and propositional formulas over σ is defined by structural induction:*

- $I \models p$ *if* p *is an atom and* $p \in I$,
- $I \models \psi \wedge \phi$ *if* $I \models \psi$ *and* $I \models \phi$,
- $I \models \psi \vee \phi$ *if* $I \models \psi$ *or* $I \models \phi$,
- $I \models \neg\psi$ *if* $I \not\models \psi$.

Figure 1.2: Definition of satisfaction

most common forms of inductive definitions in mathematics: monotone ones, respectively definitions by induction over a well-founded order. As seen in these figures, both are frequently represented as a set of informal rules. These two forms of inductive definitions are generalized by the concept of *iterated* inductive definitions (IID) [29]. Inductive definitions define their concept by describing how to *construct* it through a process of iterated application of rules starting from the empty set. Definitions by induction over a well-founded order are frequently non-monotone, as illustrated by the non-monotone rule "$I \models \neg\psi$ if $I \not\models \psi$" which derives the satisfaction of $\neg\psi$ given the non-satisfaction of ψ.

Of course, a definition is not just a set of material implications. Thus, a sensible scientific research question is to design a uniform, rule-based formalism for representing these forms of definitions. Such a study is not only useful as a formal logic study of the concept of inductive definition but it contributes to the understanding of rule-based systems and thus, to the study of the (formal and informal) semantics of logic programming and the integration of classical logic-based and rule-based approaches to knowledge representation.

Iterated inductive definitions have been studied in mathematical logic [29] but the formalisms there are not rule-based and require an extremely tedious encoding of rules and well-founded orderings into one complex formula [55]. In several papers [49, 51, 55], it was argued that, although unintended by its inventors, the rule-based formalism of logic programming under the well-founded semantics

[72] and its extension to rules with FO-bodies in [71] correctly formalizes the above mentioned forms of inductive definitions. Stated differently, if we express an informal inductive definition of one of the above kinds into a set of formal rules

$$\forall \bar{x}(P(\bar{t}) \leftarrow \phi),$$

then the informal semantics of the original definition matches the well-founded semantics of the formal rule set. E.g., in a well-founded model of the following "literal" translation of the definition in Figure 1.1:

$$\left\{ \begin{array}{c} \forall x, y \ (T_G(x, y) \leftarrow G(x, y)) \\ \forall x, y \ (T_G(x, z) \leftarrow (\exists z \ T_G(x, y) \wedge T_G(y, z))) \end{array} \right\},$$

T_G is interpreted as the transitive closure of the graph interpreting G. A similar claim holds for the literal translation of the definition of $\models$ in Figure 1.2. Thus, the rule formalism under the well-founded semantics provides the desired uniform syntax and semantics for representing the above mentioned forms of inductive definition construct.

From a KR point of view, there are several good arguments to integrate the above inductive definition construct (and hence, this generalized form of logic programming under the well-founded semantics) into FO.

1. FO and definitions are complementary KR languages: FO is a base language very suitable for expressing propositions, assertions or constraints while it is well-known that, in general, inductive definitions cannot be expressed in FO [104].

2. Definitions are important for KR. In the case of *non-inductive* definitions, their use for defining terminology was argued long time ago in Brachman and Levesque's seminal paper [23] and was the motivation for developing description logics [9]. As for *inductive* definitions, they are quite likely as important to declarative Knowledge Representation as recursive functions and procedures are to programming. Applications of inductive definitions abound in KR: various instances of transitive closure, definitions of recursive types and of concepts defined over recursive types, descriptions of dynamic worlds through definitions of states in terms of past states and effects of actions, etc. In [54], a formalization of the situation calculus in terms of iterated inductive definitions in FO(ID) yields an elegant and very general solution for the ramification problem in the context of the situation calculus.

3. Inductive definitions are also an interesting *Non-Monotonic Reasoning* language construct. A logic is non-monotonic if adding new expressions to a theory may invalidate previous inferences. Obviously, adding a new rule to an inductive definition defines a different set and hence, this operation may

invalidate previous inferences[1]. One of the main non-monotonic reasoning principles is the Closed World Assumption (CWA) [148]. The intuition underlying CWA is that "an atom is false unless it can be proven". This matches with an inductive definition in which a defined atom $P(\bar{t})$ is false unless it is explicitly derived by one of its rules $P(\bar{t}) \leftarrow \psi$ during the construction process. Hence, inductive definitions can be viewed as a very precise and well-understood form of Closed World Assumption. Moreover, it is well-known that rule formalisms under CWA can be used to represent many useful forms of *defaults*. The correspondence between CWA and inductive definition construct implies that the methodologies to represent defaults developed in, e.g., logic programming, can be used in an inductive definition formalism as well. Domain Closure [124] is another important non-monotonic reasoning principle that can be expressed through inductive definitions [55].

All the above provides a strong motivation for adding inductive definitions to FO. Thus, the resulting logic FO(ID) (or ID-Logic) extends FO not only with an inductive definition construct but also with an expressive and precise non-monotonic reasoning principle. Not surprisingly, the logic FO(ID) is strongly tied to many other logics. It is an extension of FO with inductive definitions and a conceptually clean integration of FO and LP. It integrates monotonic and non-monotonic logics. The inductive definition construct of FO(ID) formally generalizes Datalog [1]: this is a natural match, given that Datalog programs aim to *define* queries and views. FO(ID) is also strongly related to fixpoint logics. Monotone definitions in FO(ID) are a different -rule-based- syntactic sugar of the fixpoint formulas of Least Fixpoint Logic (LFP) [139, 142]. Last but not least, FO(ID), being a conceptually clean, well-founded integration of rules into classical logic, might play a unifying role in the current attempts of extending FO-based description logics with rules [177]. It thus appears that FO(ID) occupies quite a central position in the spectrum of computational and knowledge representation logics.

Several attempts to build inference systems for FO(ID) are underway. One line of research is the development of the automated reasoning method *finite model generation* for FO(ID) [116, 115, 118, 182]. They have similar applications and speed as current Answer Set Programming solvers [69, 57].

Another important topic of research is the development of deductive inference methods for FO(ID). In the first part of the thesis, we study this more traditional form of inference: *deduction*. While there remain formidable obstacles in the way of successful automated theorem proving for FO(ID), we believe that it is potentially useful as well as interesting to gain a clearer picture of the main relevant proof

[1]Observe that the concept of (non-)monotonicity is used here in two different ways. Adding a rule to a *monotone inductive definition* is a *non-monotonic reasoning* operation.

principles for FO(ID) from a proof-theoretic standpoint. Accordingly, the goal of the first part of this thesis is to develop proof foundations for reasoning in FO(ID).

Fixpoint constructs, namely the least fixpoint constructs and the greatest fixpoint constructs, are important and very successful tools in various areas in computer science, philosophy, mathematics etc. They are being used—in various shapes and forms—in order to model and to reason about knowledge. The lack of expressive power of first-order logic to represent recursion and corecursion has motivated its extension with fixpoint constructs. For example, the *μ-calculus* [97, 170], a class of temporal logics with explicit fixpoint constructs, provides a conceptually advantageous framework for specifying and reasoning about Real-Time Systems [61]. In the context of databases, query languages have been extended with fixpoint constructs to represent inductively definable concepts [19, 3]. Also, description logics have been extended with such fixpoint constructs [33]. These logics can be viewed as well-behaved fragments of first-order logic with fixpoints [139, 142]. The availability of explicit least and greatest fixpoint constructs in these logics allows inductive and coinductive concepts to be expressed in a natural way. Also in these logics, several forms of induction have been modelled: e.g., monotone induction, partial fixpoint induction, inflationary fixpoint induction.

Coinduction is the constructive principle complementary to (monotone) induction: a set can be constructed by coinduction if and only if its complement can be constructed by induction. Equivalently, a (monotone) induction corresponds to least fixpoint construction while a coinduction corresponds to a greatest fixpoint construction. Coinduction has been known to mathematicians for many years in the field of universal algebra and category theory. However, the use of formal coinductive proof techniques in computer science is a relatively new trend starting with work on process algebras [127], concurrency [144], and programming language semantics [128].

In the second part of this thesis, we extend first-order logic with nested induction and coinduction. Inspired by FO(ID), we integrate LP-style rules into fixpoint constructs and thus, the resulting fixpoint expressions are in a rule-based formalism. A rule-based fixpoint expression is called a *fixpoint definition*. We extend first-order (FO) logic with fixpoint definitions (FDs) to develop a new logic FO(FD). The fixpoint definition is a syntactic variant of a standard fixpoint construct and differs from it in two (minor) ways: an LP-like rule notation is used for expressing inductive and coinductive definitions, and the definitions define predicate constants rather than predicate variables. This modification facilitates the comparison and integration with FO(ID) and often improves knowledge representation.

An important research topic is the development of finite model generation for FO(FD), as that for FO(ID). In the current state of the art, finite model generation

problems are often solved in two phases:

1. a *grounding* phase, in which a given first-order theory and a finite domain are reduced to an equivalent variable-free, or *propositional*, theory;
2. a *propositional model generation* phase, in which the models of a given propositional theory are computed.

The second part of this thesis has the second phase, *propositional model generation for FO(FD)*, as its main topic.

The propositional fragment of FO is called *propositional calculus* (PC); that of FO(FD) is called PC(FD). The model generation problem for PC is in practice equivalent to the *satisfiability* problem for PC, which is called SAT. Likewise, the model generation problem for PC(FD) (in other words, the propositional model generation problem for FO(FD)) is called SAT(FD). Thus, the second part of the thesis seeks to develop techniques to solve the SAT(FD) problem.

1.2 Contributions

The first part of the thesis is a study of deductive inference methods of FO(ID).

In proof theory, Gentzen's sequent calculus **LK** [74, 172] is a widely known proof system for first-order logic. The sequent calculus is well-suited to a goal-directed approach for constructing logical derivations. The advantage of the method is its theoretical elegance and the fact that it focuses the proof search, with applicable proof rules constrained by logical connectives appearing in the current goal sequent. We develop a formal proof system **LPC(ID)** for PC(ID), a propositional fragment of FO(ID), based on the propositional part of Gentzen's sequent calculus. **LPC(ID)** integrates inference rules for propositional calculus and propositional inductive definitions. We present the soundness and completeness of **LPC(ID)** with respect to a slightly restricted fragment of PC(ID). We present that cut-elimination does not hold in **LPC(ID)** by showing a counter-example. We also provide some complexity results for PC(ID).

Next we study a deductive inference method for FO(ID) based on stratified least fixpoint logic [40] by defining a new logic FO(ID,SLFP) and introducing a sequent calculus for FO(ID,SLFP). We show that the resulting deductive system **LFO(ID,SLFP)** is sound for a slightly restricted fragment of FO(ID,SLFP), where all definitions have to be total. Also, we investigate a more restricted fragment of FO(ID) and show the completeness result of **LFO(ID,SLFP)** for this fragment.

The second part of the thesis is to extend first-order logic with nested induction and coinduction.

We introduce a new concept of *fixpoint definitions (FDs)*, which is a reformulation of fixpoint constructs by applying rule-based syntactic sugar. The fixpoint definitions use the format of LP-like rules which will enable us more easily to link it with logic programming notation and the notation for inductive definitions in FO(ID). We define the logic FO(FD), which is an extension of first-order logic with fixpoint definitions. In the resulting logic, almost all kinds of inductions can be expressed as well. We investigate the connections between the fixpoint definitions and non-monotone inductive definitions in FO(ID) by presenting equivalence preserving transformations of non-monotone inductive definitions to fixpoint definitions. It turns out that all kinds of inductive definitions in FO(ID) can be expressed by fixpoint definitions. Meanwhile, due to the allowance of greatest fixpoints in FO(FD), coinduction can be represented in FO(FD). Thus, some concepts, e.g., infinite structures and corecursion [16], which cannot be defined in FO(ID) through the well-founded way, can be handled naturally in FO(FD). We show that FO(FD) is strictly more expressive than FO(ID) in the context of infinite structures. We also make a comparison of FO(FD) and coinductive logic programming (see, e.g., [81, 164]).

The main topic in the second part of the thesis is to develop techniques to solve the SAT(FD) problem.

We develop an algorithm for the computation of propositional fixpoint definitions. This is based on the algorithm for the evaluation of fixpoint expressions in [27]. We give a complexity result about the SAT(FD) problem. We provide an alternative characterization of the semantics of propositional fixpoint definitions. This is based on *justification semantics* [52]. This semantics will prove essential for developing the algorithms later on in the thesis.

Based on these results, we develop algorithms to solve the SAT(FD) problem. More precisely, we develop algorithms for reasoning on PC(FD) theories, and study how to integrate them in current SAT solvers. We implement a prototype SAT(FD) solver, called MiniSat(FD), as an extension of the SAT solver MiniSat [60].

We summarize our contributions:

1. a proof system **LPC(ID)** for PC(ID);
2. a proof system **LFO(ID,SLFP)** for FO(ID) based on least fixpoint logic;
3. a new logic FO(FD) and its partial relationships with FO(ID) and coinductive logic programming;
4. a standard algorithm for the computation of fixpoint definitions;

5. a characterization of propositional fixpoint definitions using justifications;
6. several algorithms to solve the SAT(FD) problem;
7. an implementation of these algorithms;

1.3 Structure of the text

The text is structured as follows:

- In Chapter 2 we introduce the background required for the rest of the text: classical logical formalisms, logic programming under the well-founded semantics and stable model semantics, FO(ID), and Gentzen's sequent calculus.
- In Chapter 3 we develop a formal proof system **LPC(ID)** for PC(ID). We present the soundness and completeness results of **LPC(ID)** with respect to a slightly restricted fragment of PC(ID). We show that cut-elimination does not hold in **LPC(ID)** by providing a counter-example. We also provide some complexity results of PC(ID). Some of the contributions in this chapter were published in [86].
- In Chapter 4 we define a new logic FO(ID,SLFP), which is an integration of FO(ID) with stratified least fixpoint logic. We develop a formal proof system **LFO(ID,SLFP)** for FO(ID,SLFP). We present that **LFO(ID,SLFP)** is sound with respect to a slightly restricted fragment of FO(ID,SLFP). We investigate a fragment of FO(ID) and prove that **LFO(ID,SLFP)** is complete to this fragment of FO(ID). Most contributions in this chapter have been published in [84].
- In Chapter 5 we introduce a brief history of fixpoints in computer science. We present the concept of fixpoint definitions. We extend the first-order logic with fixpoint definitions to obtain a new logic FO(FD). We illustrate the relationship between FO(FD) and FO(ID) by presenting transformations from inductive definitions to alternating fixpoint definitions and showing that FO(FD) is a strictly more expressive formalism than FO(ID) on infinite structures. A comparison of FO(FD) and coinductive logic programming is also provided. The contributions in this chapter regarding the logic FO(FD) and the transformations from non-monotone inductive definitions in FO(ID) to fixpoint definitions have been published in [85].
- In Chapter 6 we present an algorithm for the computation of fixpoint definitions. We give a complexity result for the SAT(FD) problem. We provide a justification semantics of propositional fixpoint definitions. We

develop algorithms for the SAT(FD) problems and study how to integrate them in current SAT solvers. We implement a prototype SAT(FD) solver MiniSat(FD). We present a model expansion solver for FO(FD), MX(FO(FD)) and evaluate its performance using a benchmark from the model checking domain. The contributions in this chapter regarding justification semantics have been published in [85]. All other contributions have not been published before.

- Finally, we summarize the results of this thesis and conclude in Chapter 7.

Chapter 2

Preliminaries

2.1 Introduction

In this chapter we introduce concepts from the domains of classical logic and of logic programming that will be used throughout the rest of thesis. In particular, we define several logics. We also introduce Gentzen's formulation of the first-order predicate calculus **LK** (logischer klassischer Kalkül), which will be used to develop efficient deduction methods for FO(ID) in the first part of the thesis.

A *logic* consists of a syntax and a semantics. The syntax is specified by defining what constitutes a well-formed statement, or theory, in the logic; the semantics is given by defining what constitutes a *model* of a given well-formed statement. A third component of a logic is informal in nature and is not always stated: its *informal semantics*. It describes how a well-formed statement in the logic should be read, and therefore how it can intuitively be understood.

The logics that we define here are classical propositional logic, classical first-order logic, logic programming under the well-founded semantics and under the stable semantics, and FO(ID).

2.2 Classical Logic Formalisms

Logical expressions allow us to manipulate syntactical objects that refer to certain aspects of a reality or domain. We will consider logics as formal languages consisting of two essential parts: one syntactical and the other semantical. The meaning of the syntactical representation is evaluated with respect to the formal

interpretation, i.e., the semantics, that is assigned to them. In this section we present the logical formalisms that are used in the remainder of the text.

2.2.1 Propositional Logic

The first step in the formulation of a logic is to make the formal language and the formal expressions and statement precise. We start with a set of distinct elements which are called *symbols*. We describe the elements of this set as follows:

- *Logical connectives*: $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\supset$ (implies).
- *Punctuation symbols*: $(,)$.
- A set Σ of *propositional atoms*.

Normally the set Σ is assumed to be finite, although Σ may also be denumerably infinite, for instance $p_1, p_2, p_3, \ldots$. Logical and punctuation symbols are fixed in every propositional language while the propositional atoms may vary. The set Σ of propositional atoms is called a *propositional vocabulary*.

The set of *propositional calculus (PC) formulas* is inductively defined as:

- a propositional atom is a PC formula;
- if φ is a PC formula, then so is $\neg\varphi$;
- if φ, ψ are PC formulas, then so are $\varphi \wedge \psi, \varphi \vee \psi$ and $\varphi \supset \psi$.

We write $\varphi \equiv \psi$, where φ and ψ are PC formulas, to abbreviate the PC formula $(\varphi \supset \psi) \wedge (\psi \supset \varphi)$.

A PC *theory* is a set of PC formulas. For a PC theory T, we say that T is a theory *over vocabulary* Σ if for each $\varphi \in T$, φ is a PC formula over Σ. A finite PC theory is equivalent to one PC formula consisting of the conjunction of all formulas of the theory.

We now define the two-valued semantics of propositional calculus by assigning *truth values* to the propositional atoms in the propositional vocabulary. We represent the truth value *true* by $\mathbf{t}$, and *false* by $\mathbf{f}$.

Definition 2.2.1 *Given a propositional vocabulary Σ, a* Σ-interpretation *(also called a* Σ-valuation*) is a function I from Σ to the set of truth values $\{\mathbf{t}, \mathbf{f}\}$.*

Observe that there are 2^n Σ-interpretations for a propositional vocabulary Σ of n propositional atoms. Note also that every propositional atom maps to one and only one of the defined truth values. Later in the chapter we will define languages for which a third value *unknown* and a forth value *inconsistent* may be assigned to atoms.

Definition 2.2.2 *Consider the set of truth values* $\{\mathbf{t}, \mathbf{f}\}$. *The* truth order $\leq$ *on this set is induced by* $\mathbf{f} \leq \mathbf{t}$. *We define the inverse* $.^{-1}$ *of the truth values in* $\{\mathbf{t}, \mathbf{f}\}$ *as* $\mathbf{f}^{-1} = \mathbf{t}$ *and* $\mathbf{t}^{-1} = \mathbf{f}$.

Definition 2.2.3 *Consider a propositional vocabulary* Σ *and a* Σ*-interpretation* $I : \Sigma \to \{\mathbf{t}, \mathbf{f}\}$. *The* Σ*-interpretation* I *can be extended to all PC formulas over* Σ *by induction on the subformula order as follows:*

- $p^I = I(p)$ *if* $p \in \Sigma$;
- $(\neg\varphi)^I = (\varphi^I)^{-1}$;
- $(\varphi \wedge \psi)^I = min_{\leq}(\{\varphi^I, \psi^I\})$;
- $(\varphi \vee \psi)^I = max_{\leq}(\{\varphi^I, \psi^I\})$;
- $(\varphi \supset \psi)^I = max_{\leq}(\{(\varphi^I)^{-1}, \psi^I\})$.

For an arbitrary PC formula φ, we say that an interpretation I *satisfies* φ, or I is a *model* of φ, denoted $I \models \varphi$, if $\varphi^I = \mathbf{t}$. We also say that an interpretation I does not satisfy φ, or I is not a model of φ, denoted $I \not\models \varphi$, if $\varphi^I = \mathbf{f}$. We also denote $\varphi^I = \mathbf{t}$ by $I(\varphi) = \mathbf{t}$, and $\varphi^I = \mathbf{f}$ by $I(\varphi) = \mathbf{f}$. We say a propositional formula φ over Σ is a *tautology* if and only if for every Σ-interpretation I: $I \models \varphi$. A propositional formula φ over Σ is *satisfiable* iff there exists a Σ-interpretation I such that $I \models \varphi$. Finally, we say a propositional formula φ is *unsatisfiable* (or *inconsistent*), if φ is not satisfiable.

The concept of interpretation in formulas can be further extended to theories. For a PC theory T over a propositional vocabulary Σ, we say that a Σ-interpretation I satisfies a PC theory T, or I is a model of T, denoted by $I \models T$, iff I satisfies every $\varphi \in T$.

With the notation of satisfiability of PC theories in order, we proceed to define the concept of *entailment* (or *logical consequence*), which semantically relates formulas and theories.

Definition 2.2.4 *Let* T *be a PC theory over* Σ *and* φ *a PC formula over* Σ. *We say* φ *is* entailed *by (or a* logical consequence of*)* T*, if for every* Σ*-interpretation* I, $I \models T$ *implies* $I \models \varphi$. *This is denoted by* $T \models \varphi$.

To check whether a formula is entailed by a theory, we need to verify the truth value of the formula with respect to those interpretations that make the original theory true.

A much-used fragment of PC is *conjunctive normal form* (CNF). A theory is said to be in CNF if it is a conjunction of *clauses*, where a clause is a disjunction of literals. Equivalently, a CNF theory is a *set* of clauses. An arbitrary PC theory T can be transformed in linear time into a CNF theory with an extended vocabulary, whose models have a one-to-one correspondence to the models of T [175]. The advantage of CNF over full PC for computational purposes is that CNF has a very simple structure.

2.2.2 First-order Logic

First-order logic is an extension of propositional logic that allows quantifications over the elements of the domain.

We assume an infinite supply of distinct symbols, which are classified as follows:

1. Logical symbols:
   (a) Punctuation symbols: $(,)$;
   (b) Logical connectives: $\neg, \wedge, \vee, \supset$;
   (c) Quantifiers: $\exists$ (*there exists*), $\forall$ (*for all*);
   (d) Binary equality symbol: $=$;
   (e) Propositional symbols: $\top$ (*true*) and $\bot$ (*false*).
2. Non-logical symbols:
   (a) denumerably many variable symbols: $x_1, x_2, \ldots$;
   (b) denumerably many constant symbols: $c_1, c_2, \ldots$;
   (c) denumerably many function symbols: $f_1, f_2, \ldots$, each with an associated arity $k > 0$;
   (d) denumerably many predicate symbols: $P_1, P_2, \ldots$, each with an associated arity $k \geq 0$.

As usual, we identify constant symbols with 0-ary function symbols and propositional symbols with predicate symbols of arity 0. Variable and constant symbols are called *object symbols*. $=$ can be viewed as a 2-ary predicate symbol.

We define a first-order vocabulary Σ as any set of non-logical symbols. We write $\bar{\sigma}$ to denote a sequence of symbols $(\sigma_1, \sigma_2, \ldots)$ or, depending on the context, simply the set of symbols $\{\sigma_1, \sigma_2, \ldots\}$. Likewise, $\bar{x}$ is used to denote a sequence or a set of variable symbols, etc..

Definition 2.2.5 (Terms) *The set of* terms *over a first-order vocabulary* Σ*, denoted by* $Terms(\Sigma)$*, is the smallest set of expressions closed under the following rules:*

1. *any object symbol* $o \in \Sigma$ *is a term;*
2. *if* $t_1, \ldots, t_n$ *are terms and* f *is an* n*-ary function symbol, where* $n \geq 1$*, then* $f(t_1, \ldots, t_n)$ *is a term.*

We write $t[x/u]$, where x is a variable symbol and t, u are terms, to denote the term obtained by substituting u for all occurrences of x in t. We write $Var(t)$ for the set of variables appearing in the term t, and write $t[x_1, \ldots, x_n]$ for a term t such that $Var(t) \subseteq \{x_1, \ldots, x_n\}$, where $x_1, \ldots, x_n$ are distinct. In this case we may write $t[t_1, \ldots, t_n]$ to denote the term obtained by substituting $t_1, \ldots, t_n$ for $x_1, \ldots, x_n$ respectively.

We now define the well-formed formulas of first-order logic.

Definition 2.2.6 (First-order formulas) *Given a first-order vocabulary* Σ*, the set of first-order formulas (FO formulas) over* Σ *is the smallest set of expressions closed under the following rules:*

1. *if* $t_1, \ldots, t_n$ *are terms of* Σ *and* P *is an* n*-ary predicate symbol in* Σ*, then* $P(t_1, \ldots, t_n)$ *is a formula;*
2. *if* t *and* u *are terms of* Σ *then* $t = u$ *is a formula;*
3. *if* φ *is a formula then so is* $\neg\varphi$*;*
4. *if* φ *and* ψ *are formulas then so are* $\varphi \wedge \psi, \varphi \vee \psi$ *and* $\varphi \supset \psi$*;*
5. *if* φ *is a formula and* x *is a variable symbol, then* $\exists x\varphi$ *and* $\forall x\varphi$ *are formulas.*

We use the following standard abbreviation: $\varphi \equiv \psi$ for $(\varphi \supset \psi) \wedge (\psi \supset \varphi)$. Any formula of the form $P(t_1, \ldots, t_n)$ or $t = u$ is called an *atomic formula*, or an *atom*. A *literal* is an atom $P(t_1, \ldots, t_n)$ or its negation $\neg P(t_1, \ldots, t_n)$, called respectively a *positive* and a *negative* literal. For a literal l, we identify $\neg\neg l$ with l. For a set of literals S, we denote by $\neg S$ the set $\{\neg l \mid l \in S\}$, and by $\widehat{S}$ the set $S \cup \neg S$. A predicate symbol P has a *negative (positive) occurrence* in a formula φ if P has an occurrence in the scope of an odd (even) number of occurrences of the negation symbol $\neg$. We write $\varphi[x/t]$, where x is a variable symbol, t is a term and φ is a formula, to denote the formula obtained by substituting t for all occurrences of x in φ.

Definition 2.2.7 (Free variables) *The set of* free variables *occurring in a formula φ, denoted by $free(\varphi)$, is defined inductively on the structure of φ as follows:*

1. $free(P(t_1, \ldots, t_n)) = \bigcup_{1 \leq i \leq n} Var(t_i)$
2. $free(t = u) = Var(t) \cup Var(u)$
3. $free(\neg\varphi) = free(\varphi)$
4. $free(\varphi \wedge \psi) = free(\varphi \vee \psi) = free(\varphi \supset \psi) = free(\varphi) \cup free(\psi)$
5. $free(\exists x\varphi) = free(\forall x\varphi) = free(\varphi) \setminus \{x\}$

An FO formula with no free variables is called an FO *closed formula* or an FO *sentence*. An FO formula or term that contains no variables is called *ground*. An FO *theory* is a set of FO sentences. For an FO theory T, we say that T is a theory *over vocabulary* Σ if for each $\varphi \in T$, T is an FO sentence over Σ. A finite FO theory is equivalent to one FO sentence consisting of the conjunction of all sentences of the theory.

Having defined the basic syntactic concepts, we define the semantic concepts.

Definition 2.2.8 (First-order structure) *Given a first-order vocabulary Σ, a* Σ-structure *(also called a* Σ-interpretation*) I consists of a non-empty set $dom(I)$ called the* domain *of I, and a mapping of each symbol σ in Σ to a value σ^I in $dom(I)$ such that*

- *it associates each object symbol $o \in \Sigma$ with a domain element $d \in dom(I)$;*
- *it associates each n-ary function symbol $f \in \Sigma$ with an n-ary function on $dom(I)$;*
- *it associates each n-ary predicate symbol $P \in \Sigma$ with an n-ary relation on $dom(I)$.*

If $\sigma \in \Sigma$ and I is a Σ-structure, we say that I interprets *σ.*

For a given vocabulary Σ and interpretation I, a *domain atom* is of the form $P(\bar{a})$, where P is an n-ary predicate symbol in Σ and $\bar{a} \in dom(I)^n$. We sometimes restrict a Σ-interpretation I to a vocabulary $\Sigma' \subset \Sigma$. This restriction, denoted $I|_{\Sigma'}$, is the Σ'-interpretation with the same domain as I, and same interpretations on all symbols of Σ'.

Let t be a term, and let I be a structure interpreting each symbol in t. We define the denotation t^I of t under I by the usual induction:

- if t is an object symbol o, then t^I is o^I, the value of o in I;
- if $t = f(t_1, \ldots, t_n)$, then $t^I := f^I(t_1^I, \ldots, t_n^I)$.

We extend the notation t^I to tuples of terms $\bar{t}$. If $\bar{x}$ is a tuple of n variables of Σ and $\bar{a} \in dom(I)^n$, we denote by $I[\bar{x}/\bar{a}]$ the interpretation that has the same domain as I, interprets $\bar{x} = (x_1, \ldots, x_n)$ by $\bar{a} = (a_1, \ldots, a_n)$, and coincides with I on all other symbols.

Definition 2.2.9 *Let I be a structure and let φ be a formula such that each free variable in φ is interpreted by I. The truth value of φ in I, denoted φ^I, is defined inductively as follows:*

- $P(\bar{t})^I = \mathbf{t}$ *if* $\bar{t}^I \in P^I$*; otherwise* $P(\bar{t})^I = \mathbf{f}$*;*
- $(\neg\varphi)^I = (\varphi^I)^{-1}$*;*
- $(\varphi \wedge \psi)^I = min_{\leq}(\varphi^I, \psi^I)$*;*
- $(\varphi \vee \psi)^I = max_{\leq}(\varphi^I, \psi^I)$*;*
- $(\varphi \supset \psi)^I = max_{\leq}(\{(\varphi^I)^{-1}, \psi^I\})$.
- $(\exists x\varphi)^I = max_{\leq}(\{\varphi^{I[x/a]} \mid a \in dom(I)\})$*;*
- $(\forall x\varphi)^I = min_{\leq}(\{\varphi^{I[x/a]} \mid a \in dom(I)\})$.

Note that the truth of a formula φ is only well-defined in a structure interpreting each free variable of φ. We say that a structure I *satisfies* a formula φ, or I is a *model* of φ, if $\varphi^I = \mathbf{t}$. This is denoted by $I \models \varphi$.

Some important concepts of propositional logic have their equivalents in first-order logic. For a theory T, $I \models T$ iff $I \models \varphi$ for every sentence $\varphi \in T$; I is said to be a *model* of T. We say T is *satisfiable* if it has at least one model. We now define the notion of logical consequence for the first-order case.

Definition 2.2.10 *Let T be a theory. A sentence φ is called a* logical consequence *of T, or T* entails *φ, denoted by $T \models \varphi$, iff φ is true in every model of T.*

This notation generalizes also the notations for *satisfiability* and *validity* for a sentence φ.

Definition 2.2.11 *A first-order sentence is* satisfiable *iff it is true in at least one first-order structure. A sentence φ is* valid*, denoted by $\models \varphi$, iff it is true in every first-order structure.*

Herbrand interpretations

In logic programming, the domain is often restricted to the *Herbrand universe*, i.e., the set of all ground (variable-free) terms of the vocabulary. An important class of structures (resp. models) used in logic programming are the so-called *Herbrand interpretations* (resp. *Herbrand models*).

Definition 2.2.12 *Let Σ be a first-order vocabulary. The* Herbrand universe $\mathcal{U}_\Sigma$ *is the set of all ground terms of Σ. The* Herbrand base $\mathcal{B}_\Sigma$ *is the set of all ground atoms of Σ. A* Herbrand interpretation $\mathcal{I}$ *based on vocabulary Σ is an interpretation such that:*

- *The domain of $\mathcal{I}$ is $\mathcal{U}_\Sigma$.*
- *For every constant c, $c^\mathcal{I}$ is interpreted as itself c.*
- *Every n-ary function symbol f is interpreted as follows:*

$$f^\mathcal{I}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \text{ with } t_i \in \mathcal{U}_\Sigma.$$

- *Every n-ary predicate symbol P is interpreted as a subset of ${\mathcal{U}_\Sigma}^n$.*

Since in every Herbrand interpretation based on a vocabulary Σ the Herbrand universe and Herbrand base remain constant, it suffices to specify a particular Herbrand interpretation by a subset of the Herbrand base.

A Herbrand interpretation that models a theory is called a *Herbrand model*.

Definition 2.2.13 *A* Herbrand model *of a theory is a Herbrand interpretation which is a model of every sentence in the theory.*

The models of a theory that contains the *Unique Names Axioms* (UNA) [149] and the *Domain Closure Assumption* (DCA) [150] are isomorphic to Herbrand interpretations.

The UNA can be expressed in FO as follows:

- a sentence $\forall\bar{x}\forall\bar{y}\ F_i(\bar{x}) \neq F_j(\bar{y})$, for all function symbols $F_i \neq F_j$ (including constants) in the vocabulary, and where $\bar{x}$ and $\bar{y}$ have the appropriate arity for F_i and F_j;
- a sentence $\forall\bar{x}\forall\bar{y}\ F_i(\bar{x}) = F_i(\bar{y}) \supset \bar{x} = \bar{y}$, for all non-constant function symbols F_i in the vocabulary, and where $\bar{x}$ and $\bar{y}$ have the appropriate arity for F_i.

The DCA expresses that every object in the domain is represented by a variable-free term of the vocabulary. In general the DCA cannot be represented in FO. An exception is when all function symbols of the vocabulary are constants, $C_1, \ldots, C_N$; it is then the sentence

$$\forall x \ (x = C_1 \vee \ldots \vee x = C_N).$$

This expresses that every object in the domain is represented by one of the constants.

2.2.3 Three-valued and Four-valued Semantics

The purpose of propositional model generation techniques is to *construct* a model. During this construction, the eventual truth value of some atoms is still unknown—initially, *every* truth value is unknown. We treat "unknown" as a third truth value, and represent it by **u**. This construction may also (temporally) lead to inconsistency: when some atom is set to both true and false. We treat "inconsistent" as a fourth truth value, and represent it by **i**. In the remainder of this section we extend the classical semantics of FO to four-valued semantics, and explain the relation between three- and four-valued semantics.

Logic programming semantics such as the well-founded semantics (see further) are inherently three-valued. Our definition of the well-founded semantics depends on the three-valued semantics presented here. The four-valued semantics we present here is due to Belnap [18], and generalizes Kleene's three-valued semantics.

A *four-valued* Σ*-interpretation* is a pair $I = (I_1, I_2)$ of two Σ-interpretations Intuitively, I_1 underestimates what is true and I_2 overestimates it. Let $P(\bar{a})$ be a domain atom. Then

- $P(\bar{a})^I = \mathbf{t}$ if both $P(\bar{a})^{I_1} = \mathbf{t}$ and $P(\bar{a})^{I_2} = \mathbf{t}$;
- $P(\bar{a})^I = \mathbf{f}$ if both $P(\bar{a})^{I_1} = \mathbf{f}$ and $P(\bar{a})^{I_2} = \mathbf{f}$;
- $P(\bar{a})^I = \mathbf{u}$ if $P(\bar{a})^{I_1} = \mathbf{f}$ and $P(\bar{a})^{I_2} = \mathbf{t}$;
- $P(\bar{a})^I = \mathbf{i}$ if $P(\bar{a})^{I_1} = \mathbf{t}$ and $P(\bar{a})^{I_2} = \mathbf{f}$.

If for every predicate symbol $P \in \Sigma$, $P^{I_1} \subseteq P^{I_2}$ holds, then $I = (I_1, I_2)$ is said to be *consistent*. We denote this by $I_1 \subseteq I_2$. A *three-valued* Σ*-interpretation* is a consistent four-valued one. Observe that if I is a three-valued interpretation, $P(\bar{a})^{I_1} = \mathbf{t}$ implies $P(\bar{a})^{I_2} = \mathbf{t}$, and $P(\bar{a})^{I_2} = \mathbf{f}$ implies $P(\bar{a})^{I_1} = \mathbf{f}$, hence $P(\bar{a})^I$ cannot be **i**.

A three-valued Σ-interpretation (I_l, I_u) *approximates* any two-valued Σ-interpretation I such that $I_l \subseteq I \subseteq I_u$. I_l is a lower bound on the approximated interpretations (anything that is true in I_l, is true in any approximated interpretation), I_u is an upper bound on the approximated interpretations (anything that is false in I_u, is false in any approximated interpretation).

Let $\mathcal{V}_4$ be a set of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$. We define the *precision order* $\leq_p$ on $\mathcal{V}_4$ as follows:

$$\mathbf{u} \leq_p \mathbf{f} \leq_p \mathbf{i} \quad \text{and} \quad \mathbf{u} \leq_p \mathbf{t} \leq_p \mathbf{i}.$$

We define the *truth order* $\leq$ on $\mathcal{V}_4$ is defined as follows:

$$\mathbf{f} \leq \mathbf{u} \leq \mathbf{t} \quad \text{and} \quad \mathbf{f} \leq \mathbf{i} \leq \mathbf{t}.$$

Finally let us define the inverse $.^{-1}$ of the truth values in $\mathcal{V}_4$ as follows:

$$\mathbf{t}^{-1} = \mathbf{f};\ \mathbf{f}^{-1} = \mathbf{t};\ \mathbf{u}^{-1} = \mathbf{u};\ \mathbf{i}^{-1} = \mathbf{i}.$$

Definition 2.2.14 *Let I be a four-valued Σ-interpretation. The truth value of formula φ in I, denoted by φ^I, is defined inductively as follows:*

- $P(\bar{t})^I = P^I(\bar{t}^I)$;
- $(\neg\varphi)^I = (\varphi^I)^{-1}$;
- $(\varphi \wedge \psi)^I = min_{\leq}(\varphi^I, \psi^I)$;
- $(\varphi \vee \psi)^I = max_{\leq}(\varphi^I, \psi^I)$;
- $(\varphi \supset \psi)^I = max_{\leq}((\varphi^I)^{-1}, \psi^I)$;
- $(\exists x\varphi)^I = max_{\leq}(\{\varphi^{I[x/a]} \mid a \in dom(I)\})$;
- $(\forall x\varphi)^I = min_{\leq}(\{(\varphi^{I[x/a]} \mid a \in dom(I)\})$.

Definition 2.2.15 *Let $I = (I_1, I_2)$ be a four-valued interpretation and φ a formula. We define I* satisfies *φ, denoted $I \models_4 \varphi$, by induction over the size of φ as:*

- $I \models_4 P(\bar{t})$ *iff* $\bar{t}^{I_1} \in P^{I_1}$;
- $I \models_4 \neg\varphi$ *iff* $(I_2, I_1) \not\models_4 \varphi$;
- $I \models_4 \varphi \wedge \psi$ *iff* $I \models_4 \varphi$ *and* $I \models_4 \psi$;
- $I \models_4 \varphi \vee \psi$ *iff* $I \models_4 \varphi$ *or* $I \models_4 \psi$;

- $I \models_4 \varphi \supset \psi$ *iff* $I \models_4 \neg\varphi$ *or* $I \models_4 \psi$*;*
- $I \models_4 \exists x\varphi$ *iff there exists a* $d \in dom(I)$ *such that* $(I_1[x/d], I_2[x/d]) \models_4 \varphi$*;*
- $I \models_4 \forall x\varphi$ *iff for each* $d \in dom(I)$*, it holds that* $(I_1[x/d], I_2[x/d]) \models_4 \varphi$*.*

Observe that the two-valued satisfaction relation can be defined by means of the four-valued one: for a two-valued interpretation I and a formula φ, we have that $I \models \varphi$ iff $(I, I) \models_4 \varphi$.

Let I be a four-valued interpretation, φ a formula. We now have the following correspondence between φ^I and the satisfaction relation $\models_4$.

- $\varphi^I = \mathbf{t}$, meaning that φ is *certainly true* in I, if $I \models_4 \varphi$ and $I \not\models_4 \neg\varphi$;
- $\varphi^I = \mathbf{f}$, meaning that φ is *certainly false* in I, if $I \models_4 \neg\varphi$ and $I \not\models_4 \varphi$;
- $\varphi^I = \mathbf{u}$, meaning that φ's value is unknown in I, if $I \not\models_4 \varphi$ and $I \not\models_4 \neg\varphi$;
- $\varphi^I = \mathbf{i}$, meaning that φ's value is inconsistent in I, if $I \models_4 \varphi$ and $I \models_4 \neg\varphi$.

When I is a three-valued interpretation, $I \models_4 \varphi$ implies $\varphi^I = \mathbf{t}$ and $I \models_4 \neg\varphi$ implies $\varphi^I = \mathbf{f}$. Note that $I \models_4 \neg\varphi$ implies $I \not\models_4 \varphi$, but vice versa: it is possible that both $I \not\models_4 \varphi$ and $I \not\models_4 \neg\varphi$ hold, then $\varphi^I = \mathbf{u}$. Also, the following result relates the $\models_4$ relation for three-valued interpretations to the $\models$ relation for two-valued interpretations.

Proposition 2.2.16 *Let* $I = (I_l, I_u)$ *be a three-valued* Σ*-interpretation and let* I' *be a two-valued* Σ*-interpretation such that* $I_l \subseteq I' \subseteq I_u$*. Let* φ *be an FO formula over* Σ*. Then* $I \models_4 \varphi$ *implies* $I' \models \varphi$*, and* $I' \models \varphi$ *implies* $(I_u, I_l) \models_4 \varphi$*.*

The first part of this result says that for a three-valued Σ-interpretation, if $I \models_4 \varphi$, then φ is true in any I' approximated by I, hence is certainly true. The second part says that if φ is known to be true in some two-valued interpretation approximated by $I = (I_l, I_u)$, it is certainly true in the interpretation (I_u, I_l), which overestimates what is true, and underestimates what is false.

In practice, we will never work with inconsistent interpretations or truth values; throughout the text all four-valued interpretations will be consistent and therefore three-valued, unless explicitly mentioned.

The truth and precision orders defined on $\mathcal{V}_3$ and $\mathcal{V}_4$ induce obvious corresponding truth and precision orders on, respectively, three- and four-valued interpretations. Indeed, we can define $I \leq_p J$ iff for each domain atom $P(\bar{a})$, $P(\bar{a})^I \leq_p P(\bar{a})^J$, and similarly for $\leq$. The truth and precision orders can also be extended on first-order formulas as follows.

Proposition 2.2.17 *Let φ be a first-order formula over Σ and I, J be three-valued Σ-interpretations such that $I \leq_p J$. Then $\varphi^I \leq_p \varphi^J$.*

Proof. We prove this property by induction on the structure of φ.

- Case $\varphi = P(\bar{t})$. We require to show that $P(\bar{t})^I \leq_p P(\bar{t})^J$, which is immediately the case by definition of $I \leq_p J$.
- Case $\varphi = \neg\varphi_1$. By induction hypothesis, we have that $\varphi_1^I \leq_p \varphi_1^J$. In the case that either $\varphi_1^I = \mathbf{t}$ or $\varphi_1^I = \mathbf{f}$, because $\varphi_1^I \leq_p \varphi_1^J$, we can obtain that $\varphi_1^J = \varphi_1^I$. It follows immediately that $(\varphi_1^I)^{-1} = (\varphi_1^J)^{-1}$, and thus, it holds that $(\neg\varphi_1)^I \leq_p (\neg\varphi_1)^J$. When $\varphi_1^I = \mathbf{u}$, it is obvious that $(\neg\varphi_1)^I = \mathbf{u} \leq_p (\neg\varphi_1)^J$.
- Case $\varphi = \varphi_1 \wedge \varphi_2$. By induction hypothese, we have that $\varphi_1^I \leq_p \varphi_1^J$ and $\varphi_2^I \leq_p \varphi_2^J$. When $(\varphi_1 \wedge \varphi_2)^I = \mathbf{u}$, it is obvious that $(\varphi_1 \wedge \varphi_2)^I = \mathbf{u} \leq_p (\varphi_1 \wedge \varphi_2)^J$. In the case that $(\varphi_1 \wedge \varphi_2)^I = \mathbf{t}$, it is obvious that $\varphi_1^I = \varphi_2^I = \mathbf{t}$. Because $\varphi_1^I \leq_p \varphi_1^J$ and $\varphi_2^I \leq_p \varphi_2^J$, it is obtained that $\varphi_1^J = \varphi_2^J = \mathbf{t}$, and thus, $(\varphi_1 \wedge \varphi_2)^J = \mathbf{t}$. It follows immediately that $(\varphi_1 \wedge \varphi_2)^I = \mathbf{t} \leq_p \mathbf{t} = (\varphi_1 \wedge \varphi_2)^J$. In the case that $(\varphi_1 \wedge \varphi_2)^I = \mathbf{f}$, it holds that either $\varphi_1^I = \mathbf{f}$ or $\varphi_2^I = \mathbf{f}$. Suppose that $\varphi_1^I = \mathbf{f}$ (the other case is same). Because $\varphi_1^I \leq_p \varphi_1^J$, it is obtained that $\varphi_1^J = \mathbf{f}$, and thus, $(\varphi_1 \wedge \varphi_2)^J = \mathbf{f}$. Whence, we have that $(\varphi_1 \wedge \varphi_2)^I = \mathbf{f} \leq_p \mathbf{f} = (\varphi_1 \wedge \varphi_2)^J$.
- Case $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \supset \varphi_2$. These cases are similar to the case $\varphi = \varphi_1 \wedge \varphi_2$ above.
- Case $\varphi = \exists x\psi$. When $(\exists x\psi)^I = \mathbf{u}$, it is obvious that $(\exists x\psi)^I = \mathbf{u} \leq_p (\exists x\psi)^J$. In the case that $(\exists x\psi)^I = \mathbf{f}$, we have that $\psi^{I[x/a]} = \mathbf{f}$ for each domain element a. By the induction hypothesis, we have that $\psi^{I[x/a]} = \mathbf{f} \leq_p \psi^{J[x/a]}$ for each domain element a, and thus it holds that $\psi^{J[x/a]} = \mathbf{f}$ for each domain element a. It follows immediately that $(\exists x\psi)^J = \mathbf{f}$ and hence, $(\exists x\psi)^I \leq_p (\exists x\psi)^J$. In the case that $(\exists x\psi)^I = \mathbf{t}$, we have that $\psi^{I[x/a]} = \mathbf{t}$ for some domain element a. By induction hypothesis, we have that $\psi^{I[x/a]} = \mathbf{t} \leq_p \psi^{J[x/a]}$, and thus $\psi^{J[x/a]} = \mathbf{t}$. It follows directly that $(\exists x\psi)^J = \mathbf{t}$ and hence, $(\exists x\psi)^I \leq_p (\exists x\psi)^J$.
- Case $\varphi = \forall x\psi$. This case is similar to the case $\varphi = \exists x\psi$ above.

Proposition 2.2.18 *Let φ be a first-order formula over Σ, S_1 be a set of predicates which occur only positively in φ, and S_2 be a set of predicates which occur only negatively in φ. Let I, J be three-valued Σ-interpretations such that $P(\bar{a})^I \leq P(\bar{a})^J$ for each domain atom $P(\bar{a})$ with $P \in S_1$, $P(\bar{a})^J \leq P(\bar{a})^I$ for each*

domain atom $P(\bar{a})$ with $P \in S_2$, and $P(\bar{a})^I = P(\bar{a})^J$ for every other domain atom $P(\bar{a})$ with P occurring in φ. Then $\varphi^I \leq \varphi^J$.

Proof. We prove it by induction on the structure of φ.

- Case $\varphi = P(\bar{t})$. S_1 can be either empty or a singleton set $\{P\}$. In each case, it is trivial to show that $P(\bar{t})^I \leq P(\bar{t})^J$, as required.
- Case $\varphi = \neg\varphi_1$. Then for each $P \in S_1$, P occurs only negatively in φ_1 and for each $P \in S_2$, P occurs only positively in φ_1. Thus, by the induction hypothesis, we have that $\varphi_1^J \leq \varphi_1^I$. It follows immediately that $(\neg\varphi_1)^I = (\varphi_1^I)^{-1} \leq (\varphi_1^J)^{-1} = (\neg\varphi_1)^J$, as required.
- Case $\varphi = \varphi_1 \wedge \varphi_2$. Let S_1' be the set of all predicates P such that P occurs in φ_1 and $P \in S_1$ and let S_2' be the set of all predicates P such that P occurs in φ_1 and $P \in S_2$. It is obvious that for each $P \in S_1'$, P occurs only positively in φ_1 and for each $P \in S_2'$, P occurs only negatively in φ_1. Note that $S_1 \cap S_2 = \emptyset$. Hence, it can be easily shown that for each P such that P occurs in φ_1 and $P \notin S_1' \cup S_2'$, it holds that $P \notin S_1 \cup S_2$. Thus, we have that $P(\bar{a})^I \leq P(\bar{a})^J$ for each $P \in S_1'$ and an arbitrary tuple of domain elements $\bar{a}$, $P(\bar{a})^J \leq P(\bar{a})^I$ for each $P \in S_2'$ and an arbitrary tuple of domain elements $\bar{a}$, and $P(\bar{a})^I = P(\bar{a})^J$ for every other predicate P occurring in φ_1. Hence, by the hypothesis induction, it is obtained that $\varphi_1^I \leq \varphi_1^J$. Similarly, we have that $\varphi_2^I \leq \varphi_2^J$. It follows directly that $(\varphi_1 \wedge \varphi_2)^I = min_{\leq}(\varphi_1^I, \varphi_2^I) \leq min_{\leq}(\varphi_1^J, \varphi_2^J) = (\varphi_1 \wedge \varphi_2)^J$, as required.
- Case $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \supset \varphi_2$. These cases are similar to the previous case $\varphi = \varphi_1 \wedge \varphi_2$.
- Case $\varphi = \exists x\psi$. By the induction hypothesis, it is obtained that $\psi^{I[x/a]} \leq \psi^{J[x/a]}$ for every domain element a. It follows directly that $(\exists x\psi)^I = max_{\leq}(\{\psi^{I[x/a]}\}) \leq max_{\leq}(\{\psi^{J[x/a]}\}) = (\exists x\psi)^J$, as required.
- Case $\varphi = \forall x\psi$. This case is similar to the above case $\varphi = \exists x\psi$.

The above properties about the precision and truth order on the first-order formulas (and PC formulas) will be applied frequently in this text. For convenience, we will not mention them particularly for the remainder of this thesis.

2.3 Logic Programming

Problem-solving in standard programming languages, like C++ or Java, requires that the programmer supplies the necessary algorithms to solve the problem. In

other words, the programmer needs to explain how to reason about the problem. The central idea behind logic programming languages is to supply a description of the problem to a logic interpreter. The internal inference mechanism of this interpreter will do all the rest. These logic programming languages use a very simple yet powerful syntax based on extended "if-then" rules. When the conditions are satisfied, the conclusion has to be fulfilled as well.

Definition 2.3.1 *A* rule *is an expression of the form*

$$P(t_1, \ldots, t_n) \leftarrow \varphi$$

where P is a defined predicate, $t_1, \ldots, t_n$ are terms and φ is a first-order formula.

We call the atom $P(t_1, \ldots, t_n)$ the *head* of the rule and the formula φ the *body*. A rule with empty head is called an *constraint*. A rule with empty body is called a *fact*. A *logic program* is a finite set of rules. A *definite logic program* is a logic program in which the body of each rule contains only positive literals. If the body contains negative literals, it is called a *normal* rule. A logic program is called *normal* if it contains normal rules.

A program is called *ground* when it contains no variables. A ground program is called a *propositional program*.

Definition 2.3.2 *Let P be a definite logic program over a first-order vocabulary Σ. The set of all ground terms appearing in P is called the* Herbrand universe*, denoted $\mathcal{U}_P$. The* Herbrand base *$\mathcal{B}_P$ of P is the set containing all ground atoms that can be constructed from the predicates in P and the terms in $\mathcal{U}_P$.*

Let r be a rule in a program P and let σ be a mapping from variables occurring in r to the terms of $\mathcal{U}_P$. A *ground instance* of the rule r is obtained from r by replacing every variable x in r by $\sigma(x)$. The *ground version* of a program P, denoted as $ground(P)$, is obtained by replacing each rule by all its ground instances.

Example 2.3.3 *Consider the following logic program P:*

$$\left\{ \begin{array}{c} Vegetable(x) \leftarrow Tree(x) \\ Tree(Pine) \\ Vegetable(Rafflesia) \end{array} \right\}$$

The Herbrand universe $\mathcal{U}_P = \{Pine, Rafflesia\}$. The Herbrand base $\mathcal{B}_P = \{Vegetable(Pine), Vegetable(Rafflesia), Tree(Pine), Tree(Rafflesia)\}$.

$$ground(P) = \left\{ \begin{array}{c} Vegetable(Pine) \leftarrow Tree(Pine) \\ Vegetable(Rafflesia) \leftarrow Tree(Rafflesia) \\ Tree(Pine) \\ Vegetable(Rafflesia) \end{array} \right\}.$$

In the remainder of this section we will assume that all programs are already grounded. This will have no effect on the results, but simplifies notations.

Definition 2.3.4 *Let P be a definite logic program over vocabulary Σ. An* interpretation *of P is a subset $I \subseteq \mathcal{B}_P$ of ground atoms that are true. The set of all interpretations of P is denoted by $\mathcal{I}_P$.*

Definition 2.3.5 *Let P be a definite logic program over Σ. The model obtained from the intersection of all models for P is called the* least Herbrand model *or* minimal model.

The least Herbrand model of a definite logic program can be constructed by iterating on the *immediate consequence operator*:

Definition 2.3.6 *Let P be a definite logic program over Σ and $I \subseteq \mathcal{B}_P$. We define the* immediate consequence operator $\mathcal{T}_P : 2^{\mathcal{B}_P} \mapsto 2^{\mathcal{B}_P}$ *as:*

$$\mathcal{T}_P(I) = \{a \mid a \leftarrow A \in P \wedge A^I = \mathbf{t}\}.$$

Because the program is definite, this operator is monotonic in $2^{\mathcal{B}_P}$. This operator is an easy tool for identifying models, as demonstrated by the next theorem.

Theorem 2.3.7 *Let P be a definite logic program over Σ and let I be an interpretation. Then I is a model of P iff $\mathcal{T}_P(I) \subseteq I$.*

The least Herbrand model can also be retrieved by means of the immediate consequence operator.

Theorem 2.3.8 *Let P be a definite logic program over Σ. The unique minimal Herbrand model of P equals the least fixpoint of the operator $\mathcal{T}_P$. Furthermore, $\mathcal{T}_P^{\omega}(\emptyset) = lfp(\mathcal{T}_P)$.*

For normal logic programs that contain negation in the bodies of the rules, the least Herbrand model does not always exist. This fact has motivated the introduction of several semantics that try to capture the intuitions behind normal logic programs. The well-founded semantics and the stable model semantics are the most common ones, which are introduced in the following subsections.

2.3.1 Well-founded Semantics

The well-founded semantics was first defined for propositional normal logic programs by Van Gelder et al. [72], and generalized to logic programs by Van Gelder [71]. We deviate from these definitions and follow an alternative definition [56], where the well-founded model is constructed as the limit of a *well-founded sequence*.

Let I be a three-valued interpretation and $P(\bar{a})$ a domain atom. Then we denote by $I[P(\bar{a})/\mathbf{t}]$, respectively $I[P(\bar{a})/\mathbf{f}]$, the interpretation I' that is identical to I except that $I'(P(\bar{a})) = \mathbf{t}$, respectively $I'(P(\bar{a})) = \mathbf{f}$. We extend the notation to sets U of domain atoms.

Definition 2.3.9 (Well-founded sequence) *Let P be a logic program over Σ, I a three-valued Σ-interpretation. A* well-founded sequence for P from I *is any sequence $\langle I^i \rangle_{0 \leq i \leq n}$ of three-valued Σ-interpretations, such that $I^0 = I$, and each I^{i+1} is derived from I^i as follows:*

1. *$I^{i+1} = I^i[P(\bar{a})/\mathbf{t}]$ for some domain atom $P(\bar{a})$ such that $P(\bar{a})^{I^i} = \mathbf{u}$, and there is a rule $P(\bar{x}) \leftarrow \varphi \in P$ with $\varphi(\bar{x}/\bar{a})^{I^i} = \mathbf{t}$; or*
2. *$I^{i+1} = I^i[U/\mathbf{f}]$ for some set of domain atoms U such that for each $P(\bar{a}) \in U$ the following conditions hold:*
   - *$P(\bar{a})^{I^i} = \mathbf{u}$, and*
   - *for each rule $P(\bar{x}) \leftarrow \varphi \in P$, $\varphi(\bar{x}/\bar{a})^{I^i[U/\mathbf{f}]} = \mathbf{f}$.*

   Such a set U is called an unfounded set with respect to I_i.

Example 2.3.10 *We illustrate the concept of unfounded set on a propositional vocabulary. Let $P =$*

$$\left\{ \begin{array}{c} p \leftarrow q \\ q \leftarrow p \wedge r \end{array} \right\},$$

and consider the three-valued interpretation I with $p^I = q^I = \mathbf{u}$, $r^I = \mathbf{t}$. Then $\{p, q\}$ is an unfounded set with respect to I. We have that $I' = I[\{p, q\}/\mathbf{f}]$ is the interpretation with $p^{I'} = q^{I'} = \mathbf{f}, r^{I'} = \mathbf{t}$; and indeed, both $q^{I'} = \mathbf{f}$ and $(p \wedge r)^{I'} = \mathbf{f}$ hold, where q is the body of the only rule with p in the head, and $p \wedge r$ is the body of the only rule with q in the head.

A *terminal* well-founded sequence, or a *terminal well-founded induction*, is a well-founded sequence that cannot be extended. It is shown in [56] that, for a

given logic program P and three-valued interpretation I, any terminal well-founded sequence/induction for P from I has the same limit.

Let Σ be a vocabulary. Denote by $I_{H(\Sigma)}$ the three-valued Herbrand interpretation with $I_{H(\Sigma)}(P(\bar{a})) = \mathbf{u}$ for any domain atom $P(\bar{a})$.

Definition 2.3.11 (Well-founded model) *Let P be a logic program over Σ. Then the* well-founded model *of P is the limit of some (any) well-founded sequence for P from $I_{H(\Sigma)}$.*

Example 2.3.12 *Let* $P = \left\{ \begin{array}{l} p \leftarrow \neg p \wedge q \\ q \leftarrow p \wedge \neg q \end{array} \right\}$, *and $\Sigma = \{p, q\}$. We construct the well-founded model of P. $I_{H(\Sigma)} = \{p \mapsto \mathbf{u}, q \mapsto \mathbf{u}\}$. We construct a well-founded sequence for P from $I_{H(\Sigma)}$.*

- *We have $I^0 = I_{H(\Sigma)}$.*
- *We now apply the derivation rule (2). We have that $(\neg p \wedge q)^{I^0[\{p,q\}/\mathbf{f}]} = \mathbf{f}$ and $(p \wedge \neg q)^{I^0[\{p,q\}/\mathbf{f}]} = \mathbf{f}$ hold. Hence, $\{p, q\}$ is an unfounded set with respect to I^0, and we find $I^1 = I^0[\{p, q\}/\mathbf{f}]$. I^1 is two-valued and therefore terminal. Thus, the interpretation I^1 is a well-founded model of P.*

In general, the well-founded model may be three-valued.

Example 2.3.13 *We illustrate this on a definition with a propositional vocabulary. Let* $P = \left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}$, *and $\Sigma = \{p, q\}$. Then $I_{H(\Sigma)}$ is the well-founded model of P. Note that $p^{I_{H(\Sigma)}} = q^{I_{H(\Sigma)}} = \mathbf{u}$. Indeed, neither the first nor the second rule of a well-founded sequence can be applied on $I_{H(\Sigma)}$. Therefore, the well-founded sequence consisting of only this initial interpretation is terminal.*

2.3.2 Stable model Semantics

We now define the stable model semantics of logic programs. It is defined for propositional logic programs.

If a propositional logic program is *positive*, i.e., it has no negative literals in any rule body, then it has a *least model*. Let Σ be a propositional vocabulary, P

a positive logic program over Σ. Consider the set of Σ-interpretations $\mathcal{I} = \{I \mid$ for any $p \in \Sigma$ with $I(p) = \mathbf{t}$, $\exists(p \leftarrow \varphi) \in P$ with $I(\varphi) = \mathbf{t}\}$. Then *the least model of* P is the interpretation $I \in \mathcal{I}$ for which the set $\{p \mid I(p) = \mathbf{t}\}$ is subset-minimal.

The least model of P coincides with its well-founded model. It can be derived by first applying the first derivation rule of well-founded sequences (making domain atoms true) exhaustively, and then applying the second derivation rule on all remaining unknown atoms.

We introduce the *reduct* program transformation originally defined in [73]. This definition assumes P contains no nesting of negation; this can be easily generalized.

Definition 2.3.14 (Gelfond-Lifschitz reduct) *Let P be a propositional logic program over Σ, I a Σ-interpretation. Then the* Gelfond-Lifschitz reduct $\mathcal{GL}(P, I)$ *is the logic program obtained from P by replacing all occurrences (in rule bodies) of negative literals $\neg P$ by $\bot$ if $I(P) = \mathbf{t}$, and by $\top$ if $I(P) = \mathbf{f}$.*

Note that the Gelfond-Lifschitz reduct of any program under any interpretation is a positive logic program, and therefore has a least model.

Definition 2.3.15 (Stable model) *Let P be a propositional logic program over vocabulary Σ. Then a Σ-interpretation I is a* stable model of P *iff I is the least model of $\mathcal{GL}(P, I)$.*

Example 2.3.16 *Let P be the normal logic program*

$$\left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}.$$

Both $I_1 = \{p \mapsto \mathbf{t}, q \mapsto \mathbf{f}\}$ and $I_2 = \{p \mapsto \mathbf{f}, q \mapsto \mathbf{t}\}$ are stable models of P. For instance, we have that $\mathcal{GL}(P, I_1) =$

$$\left\{ \begin{array}{l} p \leftarrow \top \\ q \leftarrow \bot \end{array} \right\},$$

which indeed has I_1 as its least model. Note that the well-founded model of P is $\{p \mapsto \mathbf{u}, q \mapsto \mathbf{u}\}$.

Whereas the well-founded model (of a propositional logic program) is unique and may be three-valued, the stable models are not unique, and are two-valued. However, the following result establishes a strong relationship between well-founded models and stable models.

Proposition 2.3.17 *If the well-founded model of P is two-valued, then P has a unique stable model, and it coincides with the well-founded model.*

The inverse, however, is not true. Below is an example with a unique stable model which is not its well-founded model.

Example 2.3.18 *Let* $P = \left\{ \begin{array}{c} p \leftarrow \neg p \\ q \leftarrow \neg p \\ \leftarrow q \end{array} \right\}$.

Due to the constraint $\leftarrow q$, P has a unique stable model, namely $I = \{p \mapsto \mathbf{t}, q \mapsto \mathbf{f}\}$. The well-founded model of P, however, is $\{p \mapsto \mathbf{u}, q \mapsto \mathbf{u}\}$.

2.4 FO(ID)

We introduce FO(ID), or ID-Logic, the logic for which the first part of this thesis seeks to develop efficient deduction methods. It was originally introduced in [50], and further developed in [53, 55].

FO(ID) is an extension of first-order logic with a new construct for representing definitions that occur often in mathematics, but in general cannot be expressed in first-order logic.

Definition 2.4.1 *Let Σ be a vocabulary. A* definition *D over Σ is a set of rules of the form:*

$$\forall \bar{x}(P(\bar{t}) \leftarrow \varphi),$$

where φ is a first-order formula over Σ and $\bar{t}$ is a tuple of terms over Σ such that the free variables of φ and the variables of $\bar{t}$ all occur in $\bar{x}$.

In FO(ID), we call the rule symbol $\leftarrow$ *definitional implication* and we distinguish it from material implication $\supset$. A predicate appearing in the head of a rule of a definition D is called a *defined predicate* of D, any other symbol is called an *open symbol* of D. The sets of defined predicates, respectively open symbols of D, are denoted by $Def(D)$, respectively $Open(D) = \Sigma \setminus Def(D)$.

For simplicity of notation, and without loss of generality, we assume that every rule is of the form $\forall \bar{x}(P(\bar{x}) \leftarrow \varphi)$, i.e., the terms in the head and the free variables in the body are precisely the variables $\bar{x}$. This form can be obtained by replacing any non-variable term t in the head by a new variable x and replacing the body φ by $\varphi \wedge x = t$, and by subsequently replacing each rule of the form $\forall \bar{x}\bar{y}(P(\bar{x}) \leftarrow \varphi)$ by $\forall \bar{x}(P(\bar{x}) \leftarrow \exists \bar{y}\varphi)$.

An *FO(ID) formula* is a boolean combination of FO formulas and definitions. An *FO(ID) theory* is a set of FO(ID) formulas without free variables.

Example 2.4.2 *The following theory T is a well-formed FO(ID) theory, consisting of a definition and an FO sentence.*

$$D = \left\{ \begin{array}{c} \forall x, y\ (Ancestor(x, y) \leftarrow Parent(x, y)) \\ \forall x, y\ (Ancestor(x, y) \leftarrow \exists z\ Ancestor(x, z) \wedge Ancestor(z, y)) \end{array} \right\},$$

$$F = \forall x \neg Ancestor(x, x).$$

The informal meaning of D corresponds to the definition of the relation Ancestor in terms of the relations Parent and Ancestor itself in natural language. Here, the only sentence in T expresses that no person is ancestor of him/herself.

Unlike the well-founded and the stable model semantics, an inductive definition may have a model *for every given interpretation of $Open(D)$*. To define this, we begin by introducing a parameterized version of the well-founded semantics.

Definition 2.4.3 (Well-founded model extending I_O) *Let D be a definition over Σ and let I_O be an $Open(D)$-interpretation. Then define I'_O as the empty three-valued extension of I_O to $Def(D)$ (i.e., all domain atoms of $Def(D)$ are unknown). The* well-founded model of D extending I_O*, is the limit of any well-founded sequence for D from I'_O.*

Definition 2.4.4 (Model of D) *Let D be a definition over Σ and let I be a Σ-interpretation. Then I* is a model of D*, denoted $I \models D$, iff there exists an $Open(D)$-interpretation I_O such that I is the two-valued well-founded model of D extending I_O.*

We restate the original intension of FO(ID): to provide a formal syntax and semantics for definitions as they occur in mathematics (and add this to FO). It was shown in [49] that the formal semantics of definitions coincides with the semantics of definitions as occurring in mathematics. Intuitively, $Open(D)$ are the concepts that are used in the definition D and $Def(D)$ are the concepts that are being defined.

Example 2.4.5 *Consider again the inductive definition D in Example 2.4.2. We have $Def(D) = \{Ancestor\}$, $Open(D) = \{Parent\}$. Consider the $Open(D)$-interpretation I_O with $dom(I_O) = \{David, Mary, Patrick\}$ and $Parent^{I_O} = \{(David, Mary), (Mary, Patrick)\}$. It is easy to see that the well-founded model of D extending I_O is I with $dom(I) = dom(I_O)$, $Parent^I = Parent^{I_O}$ and $Ancestor^I = \{(David, Mary), (Mary, Patrick), (David, Patrick)\}$. Thus, $I \models D$.*

With the semantics of definitions, the semantics of FO(ID) is a simple extension of FO semantics.

Definition 2.4.6 *Let T be an FO(ID) theory over Σ and I a Σ-interpretation. Then I* is a model of *T, denoted $I \models T$, iff $I \models \varphi$ for every $\varphi \in T$.*

Example 2.4.7 *Example 2.4.5 continued. Let $T = \{D, \forall x \neg Ancestor(x, x)\}$. Then it is not difficult to see that the interpretation I as mentioned in Example 2.4.5 is a model of T because $I \models D$ and $I \models \forall x \neg Ancestor(x, x)$.*

We also define the entailment relation $\models$ between FO(ID) theory T and FO(ID) formula φ as it was done for first-order logic. $T \models \varphi$ if for every model I of T, $I \models \varphi$ holds.

PC(ID)

Definition 2.4.8 (PC(ID)) *The propositional fragment of FO(ID) is called* PC(ID).

Observe that PC(ID) is an extension of PC with propositional inductive definitions.

Remark 2.4.9 The semantics of propositional inductive definitions is just derived from that on the first-order level. Therefore, the intended semantics (informal meaning) of propositional inductive definitions is derived from the intended semantics of first-order definitions.

Consider for instance the definition $\{\ p \leftarrow q,\ q \leftarrow p\ \}$. Its declarative reading is "we define p, q as: p is true if q is true, and q is true if p is true". Formally, the only model of this definition is $\{p \mapsto \mathbf{f}, q \mapsto \mathbf{f}\}$.

Totality

Mathematicians often follow certain linguistic conventions when formulating inductive definitions. For instance, for non-monotone inductive definitions, they often specify that the definition is by induction *on some given well-founded order*. That is, the definition is meant to be used in a setting where that well-founded order is imposed, and is ill-defined otherwise.

Whether or not a (syntactically well-constructed) definition is (semantically) well-defined is expressed in FO(ID) using the semantical concept of *totality*. The formal definition of totality is based on the observation that if the well-founded

order over which the induction is specified is not imposed, then the definitional construction process may not be able to derive the truth or falsity of certain tuples in the defined relation—the well-founded model may be three-valued.

Definition 2.4.10 (Totality) *Let D be a definition over Σ, I a two-valued Σ-interpretation and T an FO(ID) theory over Σ. Then D is* total *in I if $I \models D$. The definition D is* total in *T iff for any Σ-interpretation M such that M is a model of T, the well-founded model of D extending $M|_{Open(D)}$ is two-valued. A definition D is* total *if for every two-valued interpretation I_O of its open symbols, the well-founded model of D extending I_O is two-valued.*

Intuitively, a definition D is a valid definition only in conjunction with a theory T such that D is total in T. We show an example of a definition that is total only in a specific structure.

Example 2.4.11 *Consider the definition of even numbers as follows:*

$$D_{even} = \left\{ \begin{array}{c} \forall x(E(x) \leftarrow x = 0) \\ \forall x(E(s(x)) \leftarrow \neg E(x)) \end{array} \right\}.$$

It is shown in [55] that this definition is total in the structure of the natural numbers. However, the definition is not total in many other structures, in particular in those where the successor function contains cycles or infinite descending chains. For example, D_{even} is not total in the structure I_O with domain $\{0, 1\}$, and $s^{I_O}(0) = 1, s^{I_O}(1) = 1$.

In practice, there are some syntactical conditions imposed on definitions, which can guarantee totality. One condition is that the definition is positive, or more general that the definition is stratified, i.e., that defined atoms can be semi-ordered such that in each rule, the head is greater or equal to each defined atom occurring positively in the body and strictly greater than each defined atom occurring negatively in the body. Another condition is that the definition is over a well-founded order, which is formalized in [55].

Totality is a fundamental property in the theory of non-monotone induction. Unfortunately, the problem of determining, for a given definition D and an FO(ID) theory T, whether D is total in T, is undecidable [154].

2.5 Sequent Calculus

In this section we shall give a proof system of the first-order logic **LK** presented in the sequent calculus style invented by Gentzen [74, 172]. The first part of this

thesis seeks to apply the sequent calculus to develop efficient deduction methods for FO(ID). We use the sequent calculus because it is by now well-established as a convenient formalism for proof-theoretic reasoning, which is our focus. It also serves as an elegant framework in which to write formal proofs; in fact, similar formalisms typically underlie automated theorem-proving tools.

We use the following notations. Let capital Greek letters $\Gamma, \Delta, \ldots$ denote (possibly empty) sets of formulas. Γ, Δ denotes $\Gamma \cup \Delta$ and Γ, φ denotes $\Gamma \cup \{\varphi\}$. By $\bigwedge \Gamma$, respectively $\bigvee \Gamma$, we denote the conjunction, respectively disjunction, of all formulas in Γ. By $\neg\Gamma$, we denote the set obtained by taking the negation of each formula in Γ. By $\Gamma \setminus \Delta$, we denote the set obtained by deleting from Γ all occurrences of formulas that occur in Δ.

A *sequent* is an expression of the form $\Gamma \to \Delta$. Γ and Δ are respectively called the *antecedent* and *succedent* of the sequent and each formula in Γ and Δ is called a *sequent formula*. We denote sequents by $S, S_1, \ldots$. A sequent $\Gamma \to \Delta$ is *valid*, denote by $\models \Gamma \to \Delta$, if every model of $\bigwedge \Gamma$ satisfies some formula in Δ. A *counter-model* for a sequent $\Gamma \to \Delta$ is an interpretation I such that $I \models \bigwedge \Gamma$ and $I \not\models \bigvee \Delta$. The sequent $\Gamma \to$ is equivalent to $\Gamma \to \bot$ and $\to \Delta$ is equivalent to $\top \to \Delta$, where $\bot, \top$ are logical constants denoting *false* and *true*, respectively. A sequent $\Gamma \to \Delta$ is a *subsequent of* $\Gamma' \to \Delta'$ if $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$.

An *inference rule* is an expression of the form:

$$\frac{S_1; \ldots; S_n}{S} \quad (n \geq 0).$$

Each S_i is called a *premise* of the inference rule, S is called *consequence*. Intuitively, an inference rule means that S can be inferred, given that all $S_1, \ldots, S_n$ are already inferred.

The *initial sequents*, or *axioms*, of **LK** are the sequents of the form:

$$\varphi, \Gamma \to \Delta, \varphi \quad \text{or} \quad \bot \to \Delta \quad \text{or} \quad \Gamma \to \top \quad \text{or} \quad \Gamma \to \Delta, t = t.$$

The inference rules of Gentzen's original sequent calculus **LK** for first-order logic consists of the structural rules, the logical rules and the equality rules. We give a version of them as follows.

Structural rules:

- Weakening rules: left: $\dfrac{\Gamma \to \Delta}{\varphi, \Gamma \to \Delta}$; right: $\dfrac{\Gamma \to \Delta}{\Gamma \to \Delta, \varphi}$.
- Contraction rules: left: $\dfrac{\varphi, \varphi, \Gamma \to \Delta}{\varphi, \Gamma \to \Delta}$; right: $\dfrac{\Gamma \to \Delta, \varphi, \varphi}{\Gamma \to \Delta, \varphi}$.

- Cut rule: $\dfrac{\Gamma \to \Delta, \varphi;\ \varphi, \Gamma \to \Delta}{\Gamma \to \Delta}$.

Logical rules:

- $\neg$ rules: left: $\dfrac{\Gamma \to \Delta, \varphi}{\neg\varphi, \Gamma \to \Delta}$; right: $\dfrac{\varphi, \Gamma \to \Delta}{\Gamma \to \Delta, \neg\varphi}$.
- $\wedge$ rules: left: $\dfrac{\varphi, \psi, \Gamma \to \Delta}{\varphi \wedge \psi, \Gamma \to \Delta}$; right: $\dfrac{\Gamma \to \Delta, \varphi;\ \Gamma \to \Delta, \psi}{\Gamma \to \Delta, \varphi \wedge \psi}$.
- $\vee$ rules: left: $\dfrac{\varphi, \Gamma \to \Delta;\ \psi, \Gamma \to \Delta}{\varphi \vee \psi, \Gamma \to \Delta}$; right: $\dfrac{\Gamma \to \Delta, \varphi, \psi}{\Gamma \to \Delta, \varphi \vee \psi}$.
- $\supset$ rules: left: $\dfrac{\Gamma \to \Delta, \varphi;\ \psi, \Gamma \to \Delta}{\varphi \supset \psi, \Gamma \to \Delta}$; right: $\dfrac{\varphi, \Gamma \to \Delta, \psi}{\Gamma \to \Delta, \varphi \supset \psi}$.

The above logical rules are called *propositional rules*.

- $\forall$ rules: left: $\dfrac{\varphi(t), \Gamma \to \Delta}{\forall x\varphi(x), \Gamma \to \Delta}$; right: $\dfrac{\Gamma \to \Delta, \varphi(x)}{\Gamma \to \Delta, \forall y\varphi(y)}$ $x \notin free(\Gamma \cup \Delta)$.
- $\exists$ rules: left: $\dfrac{\varphi(x), \Gamma \to \Delta}{\exists y\varphi(y), \Gamma \to \Delta}$ $x \notin free(\Gamma \cup \Delta)$; right: $\dfrac{\Gamma \to \Delta, \varphi(t)}{\Gamma \to \Delta, \exists x\varphi(x)}$.

Note that in an instance of any of the logical rules, the distinguished formula that is introduced in the conclusion, i.e., that does not appear in any of the premises, is called the *active formula* of the rule instance. Also, in instances of the rules right $\forall$ and left $\exists$, the "fresh" variable x used for the introduced quantifier is called the *eigenvariable* of the rule instance.

In first-order logic we treat equality $=$ as a primitive logical symbol. Accordingly, we give inference rules for introducing equality formulas on the left and right of sequents as follows.

Equality rules:

- left: $\dfrac{\varphi[x/t], \Gamma \to \Delta}{t = u, \varphi[x/u], \Gamma \to \Delta}$; right: $\dfrac{\Gamma \to \Delta, \varphi[x/t]}{t = u, \Gamma \to \Delta, \varphi[x/u]}$.

The first such rules for equality seem to have been formulated by Wang [179] in 1960, and were later investigated more fully by Kanger [95].

For convenience, our version of **LK** exhibits a minor deviation from Gentzen's original formulation, specifically, Gentzen's original exchange rules, governing the

ordering of formulas within a sequent, are rendered redundant by our use of sets in sequents. This difference is inessential and is commonly adopted for the purposes of convenience. It is easily seen that a sequent is provable in Gentzen's original formulation of **LK** just in case it is provable in our variant.

We now explain the notion of a formal proof, i.e., proof in **LK**.

Definition 2.5.1 *A* proof *(in* **LK***), or* **LK**-proof *for a sequent S, is a tree T of sequents with root S satisfying the following conditions:*

1. *Each leaf of T must be an axiom.*
2. *For each interior node S' there exists an instance of an inference rule in* **LK** *such that S' is the consequence of that instance while the children of S' are precisely the premises of that instance.*

The following terminology and conventions will be used in discussing formal proofs in **LK**.

Definition 2.5.2 *From Definition 2.5.1, it follows that there is a unique root in a proof T. A proof with root S is called a* proof ending with *S or a* proof of *S. A sequent S is called* provable *in* **LK***, or* **LK**-provable*, if there is an* **LK***-proof of it. A formula φ is called* **LK**-provable *(or a* theorem of **LK***) if the sequent $\rightarrow \varphi$ is* **LK***-provable. The prefix "***LK***-" will often be omitted from "***LK***-proof" and "***LK***-provable".*

Example 2.5.3 *The following is an* **LK***-proof.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi(a) \rightarrow \varphi(a)}{\varphi(a) \rightarrow \exists x \varphi(x)}\, \textit{right}\,\exists}{\rightarrow \exists x \varphi(x), \neg \varphi(a)}\, \textit{right}\,\neg}{\rightarrow \exists x \varphi(x), \forall y \neg \varphi(y)}\, \textit{right}\,\forall}{\neg \forall y \neg \varphi(y) \rightarrow \exists x \varphi(x)}\, \textit{left}\,\neg}{\rightarrow \neg \forall y \neg \varphi(y) \supset \exists x \varphi(x)}\, \textit{right}\,\supset$$

It should be noted that the consequence of right $\forall$ does not contain the eigenvariable a.

Theorem 2.5.4 (Soundness and Completeness) *$\Gamma \rightarrow \Delta$ is provable in* **LK** *if and only if $\models \Gamma \rightarrow \Delta$.*

Remark 2.5.5 *The above theorem connects proof theory with semantics. The "only if" part of the theorem is the statement of the soundness of* **LK**. *Soundness means: all provable sequents are valid. The soundness theorem is easily proved by induction on number of inferences in a proof of the sequent. The "if" part of the theorem is the statement of the completeness of* **LK**. *In general, a system is said to be complete if and only if every valid formula is provable in the system. Gödel showed the completeness with respect to postulates of predicate logic deriving from the Principia Mathematica [92]. The completeness theorem for* **LK** *certainly is a consequence of Gödel's result.*

Chapter 3

LPC(ID): a proof system for PC(ID)

3.1 Introduction

In this chapter, we formulate a proof system, **LPC(ID)**, for the logic PC(ID) in the sequent calculus style originally developed by Gentzen [74, 172]. Our system can be seen essentially as a propositional part of classical sequent calculus adaptation of inference rules for propositional inductive definitions.

We introduce a normal form of inductive definitions in Section 3.2. In Section 3.3, we give the proof rules of **LPC(ID)**, which are the rules of Gentzen's original sequent calculus for propositional logic, augmented with rules for introducing defined atoms on the left and right of sequents, a rule for inferring the non-totality of definitions and a rule for introducing definitions on the right of sequents. In Section 3.4, 3.5 we show that our proof system is sound and complete for a slightly restricted fragment of PC(ID). We present that cut-elimination does not hold in our proof system by showing a counter-example in Section 3.6, and establish some simple complexity results about satisfiability problem for PC(ID) in Section 3.7. We finish with conclusions and related work in Section 3.8.

By developing a proof system for PC(ID), we want to enhance the understanding of proof-theoretic foundations of FO(ID). The proof-theoretic perspective also allows us to investigate the possibility of FO(ID) as an assertion language for program verification, which can overcome the limitations of first order logic.

Another application of this work could be the development of tools to check the correctness of the outputs generated by PC(ID) model generators such

as MIDL [115] and MINISAT(ID) [118]. Given a PC(ID) theory T as input, MINISAT(ID) outputs a model for T or concludes that T is unsatisfiable. In the former case, an independent *model checker* can be used to check whether the output is indeed a model of T. However, when MINISAT(ID) concludes that T is unsatisfiable, it is less obvious how to check the correctness of this answer. One solution is to transform a trace of MINISAT(ID)'s computation into a proof of unsatisfiability in some PC(ID) proof system. An independent *proof checker* can be used to check this formal proof. Model and proof checkers can be a great help to detect bugs in model generators. An analogous checker for the Boolean Satisfiability problem (SAT) solvers was described in [184].

3.2 Preliminaries

Recall that a literal is an atom P or its negation $\neg P$. An atom P has a negative (positive) occurrence in formula φ if P has an occurrence in the scope of an odd (even) number of occurrences of the negation symbol $\neg$ in φ.

An arbitrary propositional definition D can be transformed into an equivalent definition such that each $P \in Def(D)$ is defined by exactly one rule, denoted by $P \leftarrow \varphi_P$. We call $P \leftarrow \varphi_P$ *the rule defining* P. φ_P will be used to denote the body of the rule defining P. The transformation is obtained by merging all rules with the same head atom into one rule as follows: $P \leftarrow \varphi_1, \ldots, P \leftarrow \varphi_n$ becomes $P \leftarrow \varphi_1 \vee \ldots \vee \varphi_n$, for all defined atoms P, and thus $\varphi_P = \varphi_1 \vee \ldots \vee \varphi_n$. Without loss of generality, in what follows we assume that there is only one rule $P \leftarrow \varphi_P$ in a definition D for every $P \in Def(D)$.

Example 3.2.1 *The following definition*

$$\left\{ \begin{array}{c} P \leftarrow O_1 \wedge Q \\ P \leftarrow P \\ Q \leftarrow Q \wedge P \\ Q \leftarrow O_2 \end{array} \right\}$$

is equivalent to this one:

$$\left\{ \begin{array}{l} P \leftarrow (O_1 \wedge Q) \vee P \\ Q \leftarrow (Q \wedge P) \vee O_2 \end{array} \right\}.$$

Given two disjunct propositional vocabularies Σ and Σ', a Σ-interpretation I and a Σ'-interpretation I', the $\Sigma \cup \Sigma'$-interpretation mapping each element P of Σ to P^I and each $P \in \Sigma'$ to $P^{I'}$ is denoted by $I + I'$. When $\Sigma' \subseteq \Sigma$, we denote the restriction of a Σ-interpretation I to the symbols of Σ' by $I|_{\Sigma'}$. For a Σ-interpretation I, a truth value v and an atom $P \in \Sigma$, we denote by $I[P/v]$ the

Σ-interpretation that assigns v to P and corresponds to I for all other atoms. We extend this notation to sets of atoms, i.e., for a Σ-interpretation I, a truth value v and a set of atoms S, we denote by $I[S/v]$ the Σ-interpretation that assigns v to every $P \in S$ and corresponds to I for all other atoms.

3.3 The deductive system for PC(ID)

In this section, we shall give the inference rules of the deductive system for PC(ID), **LPC(ID)**, presented in the sequent calculus style invented by Gentzen [74, 172]. We use sequent calculus because it is by now well-established as a convenient formalism for proof-theoretic reasoning, which is our focus. It also serves as an elegant framework in which to write formal proofs. In fact, similar formalisms typically underlie automated theorem-proving tools.

The inference rules for **LPC(ID)** consist of *structural* rules, *logical* rules and *definition* rules. The structural and logical rules, which follows directly the propositional inference rules in **LK**, deal with the propositional part of PC(ID) and are as given in Chapter 2.

Our deductive system **LPC(ID)** is then obtained from the propositional part of **LK** by adding rules for introducing defined atoms on the right and left of sequents, a rule for inferring the non-totality of definitions and a rule for introducing definitions on the right of sequents. The definition rules of **LPC(ID)** consist of the *right definition* rule, the *left definition* rule, the *non-total* definition rule and the *definition introduction* rule.

Right definition rule for P.

The *right definition rule* introduces defined atoms in the succedents of sequents. It allows inferring the truth of a defined atom from a definition D and is therefore closely related to the first step for extending a well-founded induction. Let D be a definition and P a defined atom of D. The right definition rule for P is given as follows.

$$\frac{\Gamma \to \Delta, \varphi_P}{D, \Gamma \to \Delta, P}$$

where Γ and Δ are arbitrary sets of PC(ID) formulas.

We illustrate this inference rule with an example.

Example 3.3.1 *Consider the definition*

$$D = \left\{ \begin{array}{c} P \leftarrow P \wedge \neg Q \\ Q \leftarrow \neg P \end{array} \right\}.$$

The instance of the right definition rule for P is

$$\frac{\Gamma \to \Delta, P \wedge \neg Q}{D, \Gamma \to \Delta, P},$$

and the instance of the right definition rule for Q is

$$\frac{\Gamma \to \Delta, \neg P}{D, \Gamma \to \Delta, Q}.$$

Left definition rule for $P_i \in U$.

The *left definition rule* introduces defined atoms in the antecedents of sequents. It allows inferring the falsity of a defined atom from a definition D and is therefore closely related to the second step for extending a well-founded induction.

We first introduce some notations. Given a set U of atoms, let $U^{\triangleright}$ be a set consisting of one new atom $P^{\triangleright}$ for every $P \in U$. The vocabulary Σ augmented with these symbols is denoted by $\Sigma^{\triangleright}$. Given a PC formula φ, $\varphi^{\triangleright}$ denotes the formula obtained by replacing all positive occurrences of an atom $P \in U$ in φ by $P^{\triangleright}$. We call $\varphi^{\triangleright}$ the *renaming* of φ with respect to U. For a set of PC formulas F, $F^{\triangleright}$ denotes $\{\varphi^{\triangleright} \mid \varphi \in F\}$. For arbitrary PC formula φ, by $\neg\varphi^{\triangleright}$, we mean $\neg(\varphi^{\triangleright})$.

Let D be a definition over Σ and U a non-empty set of atoms such that $U \subseteq Def(D)$. Denote by $\neg U^{\triangleright}$ the set $\{\neg P^{\triangleright} | P \in U\}$. Let Γ and Δ be sets of PC(ID) formulas over Σ. The left definition rule for every $P_i \in U$ is given as follows, where $U = \{P_1, \ldots, P_n\}$.

$$\frac{\neg U^{\triangleright}, \Gamma \to \Delta, \neg\varphi^{\triangleright}_{P_1}; \ldots; \neg U^{\triangleright}, \Gamma \to \Delta, \neg\varphi^{\triangleright}_{P_n}}{P_i, D, \Gamma \to \Delta}.$$

Actually, in the left definition rule, the set of atoms U is a candidate unfounded set of D.

We illustrate this inference rule with an example.

Example 3.3.2 *Given a definition* $D = \left\{ \begin{array}{c} P \leftarrow P \wedge \neg Q \\ Q \leftarrow Q \end{array} \right\}$,

- *$U = \{P\}$, the instance of the left definition rule for $P \in U$ is*

$$\frac{\neg P^{\triangleright}, \Gamma \to \Delta, \neg(P^{\triangleright} \wedge \neg Q)}{P, D, \Gamma \to \Delta}$$

- *$U = \{Q\}$, the instance of the left definition rule for $Q \in U$ is*

$$\frac{\neg Q^{\triangleright}, \Gamma \to \Delta, \neg Q^{\triangleright}}{Q, D, \Gamma \to \Delta}$$

- *$U = \{P, Q\}$, the instance of the left definition rule for $P \in U$ is*

$$\frac{\neg P^{\triangleright}, \neg Q^{\triangleright}, \Gamma \to \Delta, \neg(P^{\triangleright} \wedge \neg Q); \quad \neg P^{\triangleright}, \neg Q^{\triangleright}, \Gamma \to \Delta, \neg Q^{\triangleright}}{P, D, \Gamma \to \Delta}$$

- *$U = \{P, Q\}$, the instance of the left definition rule for $Q \in U$ is*

$$\frac{\neg P^{\triangleright}, \neg Q^{\triangleright}, \Gamma \to \Delta, \neg(P^{\triangleright} \wedge \neg Q); \quad \neg P^{\triangleright}, \neg Q^{\triangleright}, \Gamma \to \Delta, \neg Q^{\triangleright}}{Q, D, \Gamma \to \Delta}.$$

Non-total definition rule for D.

The *non-total definition rule* allows inferring the non-totality of a definition D with respect to a PC(ID) theory.

We introduce some notations first. Let D be a definition over Σ and V a non-empty set of atoms such that $V \subseteq Def(D)$. Denote by $\Sigma^{\diamond}$ the vocabulary $\Sigma \cup V^{\triangleright} \cup V^{\diamond}$, where both $V^{\triangleright}$ and $V^{\diamond}$ are sets of new and different renamings $P^{\triangleright}$ and $P^{\diamond}$ of all symbols P of V. Denote by $\varphi^{\diamond}$ the formula obtained by replacing each positive occurrence of each $P \in V$ in φ by $P^{\triangleright}$ and each negative occurrence of each $P \in V$ in φ by $P^{\diamond}$. Denote by $D^{\diamond}$ the definition $\{P^{\triangleright} \leftarrow \varphi_P^{\diamond} \mid P \in V$ and $P \leftarrow \varphi_P \in D\}$ over the new vocabulary $\Sigma^{\diamond}$. Let Γ and Δ be sets of PC(ID) formulas over Σ. Then the non-total definition rule for D is given as follows.

$$\frac{V^{\diamond}, D^{\diamond}, \Gamma \to \Delta, \bigwedge \neg V^{\triangleright}; \ \neg V^{\diamond}, D^{\diamond}, \Gamma \to \Delta, \bigwedge V^{\triangleright}}{D, \Gamma \to \Delta}$$

We illustrate this inference rule with an example.

Example 3.3.3 *Given a definition* $D = \left\{ \begin{array}{c} P \leftarrow P \wedge \neg Q \\ Q \leftarrow \neg Q \wedge R \\ R \leftarrow \neg R \end{array} \right\}$, $V = \{Q, R\}$ *and* Γ *and* Δ *empty sets. Then the instance of the non-total definition for* D *is*

$$\frac{Q^{\diamond}, R^{\diamond}, D^{\diamond} \to \neg Q^{\triangleright} \wedge \neg R^{\triangleright}; \ \neg Q^{\diamond}, \neg R^{\diamond}, D^{\diamond} \to Q^{\triangleright} \wedge R^{\triangleright}}{D \to},$$

where $D^{\diamond} = \left\{ \begin{array}{c} Q^{\triangleright} \leftarrow \neg Q^{\diamond} \wedge R^{\triangleright} \\ R^{\triangleright} \leftarrow \neg R^{\diamond} \end{array} \right\}$.

What is the cause of the non-totality of a definition? A simple and very general syntactic criterion that guarantees that a definition is total can be phrased in terms of the dependency relation, induced by the definition. It is explained in [55] that in some examples of non-total definitions, the dependency relation of the definition contains infinite descending chains in which atoms depend negatively on the same or other atoms. When this happens, the stable operator oscillates between a structure in which all atoms of the chain are false and one in which these atoms are true.

We do not have an inference rule to prove totality of all definitions in the context of a certain set Γ of PC(ID) formulas. Such an inference rule would involve proving that each model of Γ can be extended to a model of the definition. In fact, we cannot even formulate this condition as a sequent.

Definition introduction rule for D.

The three definitional inference rules introduced so far, introduce a definition in the antecedent of the consequence. Hence, none of these rules can be used to infer that under certain conditions a definition holds. The *definition introduction rule* allows inferring the truth of a total definition from PC(ID) formulas.

We introduce some notations. Let D be a total definition. Denote by P' a new defined atom for each $P \in Def(D)$. Denote by Σ' the vocabulary $\Sigma \cup \{P' \mid P \in Def(D)\}$. Denote by D' the definition over the new vocabulary Σ' obtained by replacing each occurrence of each defined symbol P in D by P'. Let Γ and Δ be sets of PC(ID) formulas over the old vocabulary Σ. The definition introduction rule for D is given as follows, where $P_1, \ldots, P_n$ are all defined atoms of D.

$$\frac{D', \Gamma \to \Delta, P_1' \equiv P_1; \ldots; D', \Gamma \to \Delta, P_n' \equiv P_n}{\Gamma \to \Delta, D}$$

We illustrate this inference rule with an example.

Example 3.3.4 *Given a definition* $D = \left\{ \begin{array}{c} P \leftarrow O \\ Q \leftarrow Q \wedge P \end{array} \right\}$, $\Gamma = \{O, P, \neg Q\}$ *and* Δ *an empty set. Then the instance of the definition introduction rule for* D *is*

$$\frac{D', O, P, \neg Q \to P' \equiv P; \quad D', O, P, \neg Q \to Q' \equiv Q}{O, P, \neg Q \to D},$$

where $D' = \left\{ \begin{array}{c} P' \leftarrow O \\ Q' \leftarrow Q' \wedge P' \end{array} \right\}$.

The inference rule proposed here has a definition in the succedent of its premise and hence, allows to infer the truth of a definition. Unfortunately, this rule is only

sound given that the inferred definition is total. We will give an example to show that the definition introduction rule is not sound given that the inferred definition is non-total right after proving the soundness of this inference rule.

In most settings we are interested in deriving PC formulas from a PC theory and definitions. This means that definitions occur only on the left side of a sequent. As shown later, **LPC(ID)** is sound with respect to such a sequent. Moreover, in practice definitions are total, and non-total definitions are aberrant. There are general syntactical conditions that guarantee totality and apply in 99% of practical domains. **LPC(ID)** will be proved to be complete in the normal case that all definitions are total.

Proofs of LPC(ID).

We now come to the notion of an **LPC(ID)**-*proof* for a sequent.

Definition 3.3.5 *An* **LPC(ID)**-proof *for a sequent S, is a tree T of sequents with root S. Moreover, each leaf of T must be an axiom and for each interior node S' there exists an instance of an inference rule in* **LPC(ID)** *such that S' is the consequence of that instance while the children of S' are precisely the premises of that instance. T is often called a* proof tree *for S. A sequent S is called* provable *in* **LPC(ID)**, *or* **LPC(ID)**-provable, *if there is an* **LPC(ID)**-*proof for it.*

Example 3.3.6 *Given a definition*

$$D = \left\{ \begin{array}{c} P \leftarrow O \\ Q \leftarrow Q \wedge P \end{array} \right\},$$

the following is an **LPC(ID)**-*proof for $O, D \rightarrow P \wedge \neg Q$.*

$$\dfrac{\dfrac{O \rightarrow O}{O, D \rightarrow P}\ \textit{right definition rule} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Q^{\triangleright} \rightarrow Q^{\triangleright}}{\neg Q^{\triangleright}, Q^{\triangleright} \rightarrow}\ \textit{left } \neg}{\neg Q^{\triangleright}, Q^{\triangleright}, P \rightarrow}\ \textit{left weakening}}{\neg Q^{\triangleright}, Q^{\triangleright} \wedge P \rightarrow}\ \textit{left } \wedge}{\neg Q^{\triangleright} \rightarrow \neg(Q^{\triangleright} \wedge P)}\ \textit{right } \neg}{Q, D \rightarrow}\ \textit{left definition rule}}{D \rightarrow \neg Q}\ \textit{right } \neg \qquad \dfrac{}{O, D \rightarrow \neg Q}\ \textit{left weakening}}{O, D \rightarrow P \wedge \neg Q}\ \textit{right } \wedge$$

3.4 Soundness of LPC(ID)

Definition 3.4.1 *Let Σ be a propositional vocabulary and I a two-valued Σ-interpretation. Then a sequent $\Gamma \to \Delta$ is said to be* true *in I if we have $I \models \varphi$ for all $\varphi \in \Gamma$ implies $I \models \psi$ for some $\psi \in \Delta$.*

Recall that $\Gamma \to \Delta$ is valid if it is true in all interpretations.

We now prove that inference rules of **LPC(ID)** are sound, i.e., if all premises of an inference rule are valid then the consequence of that rule is valid. The main interesting cases are the definition rules. Although the proof is fairly standard in the remaining cases, we nevertheless treat them comprehensively here for the sake of completeness. We firstly give the soundness proof of every inference rule in the classical sequent calculus for propositional logic as follows.

Lemma 3.4.2 *Let $\mathcal{R}$ be an arbitrary inference rule in the sequent calculus for propositional logic and I an interpretation. If all premises of $\mathcal{R}$ are true in I, then the conclusion of $\mathcal{R}$ is also true in I.*

Proof. Of course, we just need to check that each rule in the sequent calculus for propositional logic has the desired property.

- **Case (Ax):**
$$\Gamma, \varphi \to \varphi, \Delta$$
Suppose that $I \models \bigwedge \Gamma$ and $I \models \varphi$. It follows immediately that $I \models \varphi \vee \bigvee \Delta$, as desired.

- **Case (Ax):**
$$\bot \to \Delta$$
Since there is no interpretation I such that $I \models \bot$, it therefore holds that $I \models \bigvee \Delta$, as desired.

- **Case (Ax):**
$$\Gamma \to \top$$
Suppose that $I \models \bigwedge \Gamma$. It is trivial to have that $I \models \top$, as desired.

- **Case (Left Weakening):**
$$\frac{\Gamma \to \Delta}{\varphi, \Gamma \to \Delta}$$
Suppose that $I \models \varphi$ and $I \models \bigwedge \Gamma$. Since $I \models \bigwedge \Gamma$ and since $\Gamma \to \Delta$ is true in I by assumption, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Right Weakening):**

$$\frac{\Gamma \to \Delta}{\Gamma \to \Delta, \varphi}$$

Suppose that $I \models \bigwedge \Gamma$. Since $\Gamma \to \Delta$ is true in I by the assumption, it holds that $I \models \bigvee \Delta$, and thus $I \models \bigvee \Delta \vee \varphi$, as desired.

- **Case (Left Contraction):**

$$\frac{\varphi, \varphi, \Gamma \to \Delta}{\varphi, \Gamma \to \Delta}$$

Suppose that $I \models \varphi$ and $I \models \bigwedge \Gamma$. Since $I \models \varphi$, $I \models \varphi$, $I \models \bigwedge \Gamma$, and since $\varphi, \varphi, \Gamma \to \Delta$ is true in I by the assumption, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Right Contraction):**

$$\frac{\Gamma \to \Delta, \varphi, \varphi}{\Gamma \to \Delta, \varphi}$$

Suppose that $I \models \bigwedge \Gamma$. Since $\Gamma \to \Delta, \varphi, \varphi$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, it holds that $I \models \bigvee \Delta \vee \varphi$, as desired.

- **Case (Cut):**

$$\frac{\Gamma \to \Delta, \varphi; \quad \varphi, \Gamma \to \Delta}{\Gamma \to \Delta}$$

Suppose that $I \models \bigwedge \Gamma$. Since the first premise $\Gamma \to \Delta, \varphi$ is true in I and the fact that $I \models \bigwedge \Gamma$, it holds that either $I \models \bigvee \Delta$ or $I \models \varphi$. If $I \models \bigvee \Delta$, then we are done; otherwise $I \models \varphi$ and since the second premise $\varphi, \Gamma \to \Delta$ is true in I, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Left $\neg$):**

$$\frac{\Gamma \to \Delta, \varphi}{\neg\varphi, \Gamma \to \Delta}$$

Suppose that $I \models \neg\varphi$ and $I \models \bigwedge \Gamma$. Since $\Gamma \to \Delta, \varphi$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, we have that either $I \models \varphi$ or $I \models \bigvee \Delta$. Since $I \models \neg\varphi$, it follows immediately that $I \models \bigvee \Delta$, as desired.

- **Case (Right $\neg$):**

$$\frac{\varphi, \Gamma \to \Delta}{\Gamma \to \Delta, \neg\varphi}$$

Suppose that $I \models \bigwedge \Gamma$. If $I \models \neg\varphi$, then we are done; otherwise $I \models \varphi$ and since the premise $\varphi, \Gamma \to \Delta$ is true in I and the fact that $I \models \bigwedge \Gamma$, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Left $\wedge$):**

$$\frac{\varphi, \psi, \Gamma \to \Delta}{\varphi \wedge \psi, \Gamma \to \Delta}$$

Suppose that $I \models \varphi \wedge \psi$ and $I \models \bigwedge \Gamma$. Thus it holds that $I \models \varphi$, $I \models \psi$ and $I \models \bigwedge \Gamma$. Since $\varphi, \psi, \Gamma \to \Delta$ is true in I by the assumption, we have that $I \models \bigvee \Delta$, as desired.

- **Case (Right $\wedge$):**

$$\frac{\Gamma \to \Delta, \varphi;\ \Gamma \to \Delta, \psi}{\Gamma \to \Delta, \varphi \wedge \psi}$$

Suppose that $I \models \bigwedge \Gamma$. Since the first premise $\Gamma \to \Delta, \varphi$ is true in I, it holds that either $I \models \bigvee \Delta$ or $I \models \varphi$. Since the second premise $\Gamma \to \Delta, \psi$ is true in I, it holds that either $I \models \bigvee \Delta$ or $I \models \psi$. If $I \models \bigvee \Delta$, then we are done; otherwise $I \models \varphi$ and $I \models \psi$ and it thus holds that $I \models \varphi \wedge \psi$, as desired.

- **Case (Left $\vee$):**

$$\frac{\varphi, \Gamma \to \Delta;\ \psi, \Gamma \to \Delta}{\varphi \vee \psi, \Gamma \to \Delta}$$

Suppose that $I \models \varphi \vee \psi$ and $I \models \bigwedge \Gamma$. In the case that $I \models \varphi$ and $I \models \bigwedge \Gamma$, since the first premise $\varphi, \Gamma \to \Delta$ is true in I, we have that $I \models \bigvee \Delta$, as desired. In the case that $I \models \psi$ and $I \models \bigwedge \Gamma$, since the second premise $\psi, \Gamma \to \Delta$ is true in I, it follows directly that $I \models \bigvee \Delta$, as desired.

- **Case (Right $\vee$):**

$$\frac{\Gamma \to \Delta, \varphi, \psi}{\Gamma \to \Delta, \varphi \vee \psi}$$

Suppose that $I \models \bigwedge \Gamma$. Since $\Gamma \to \Delta, \varphi, \psi$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, we have that $I \models \bigvee \Delta$ or $I \models \varphi$ or $I \models \psi$. In any case, it holds that $I \models \bigvee \Delta \vee \varphi \vee \psi$, as desired.

- **Case (Left $\supset$):**

$$\frac{\Gamma \to \Delta, \varphi;\ \psi, \Gamma \to \Delta}{\varphi \supset \psi, \Gamma \to \Delta}$$

Suppose that $I \models \varphi \supset \psi$ and $I \models \bigwedge \Gamma$, and thus $I \models \neg\varphi \vee \psi$. In the case that $I \models \neg\varphi$ and $I \models \bigwedge \Gamma$, since the first premise $\Gamma \to \Delta, \varphi$ is true in I and $I \not\models \varphi$, we have that $I \models \bigvee \Delta$, as desired. In the case that $I \models \psi$ and $I \models \bigwedge \Gamma$, since the second premise $\psi, \Gamma \to \Delta$ is true in I, it follows directly that $I \models \bigvee \Delta$, as desired.

- **Case (Right $\supset$):**

$$\frac{\varphi, \Gamma \to \Delta, \psi}{\Gamma \to \Delta, \varphi \supset \psi}$$

Suppose that $I \models \bigwedge \Gamma$. In the case that $I \models \varphi$ and $I \models \bigwedge \Gamma$, since the premise $\varphi, \Gamma \to \Delta, \psi$ is true in I, we have that $I \models \bigvee \Delta$ or $I \models \psi$. In any

case, it holds that $I \models \bigvee \Delta \vee \psi \vee \neg\varphi$, as desired. In the case that $I \not\models \varphi$ and $I \models \bigwedge \Gamma$, we have that $I \models \neg\varphi$. It follows directly that $I \models \bigvee \Delta \vee \neg\varphi \vee \psi$, as desired.

We now proceed to give the soundness proof of each definition rule.

Lemma 3.4.3 *Let I be a model of D and P a defined atom of D. $I \models P$ if and only if $I \models \varphi_P$.*

Proof. Because I is a model of D, there exists a terminal well-founded induction $(I^n)_{n\leq\xi}$ for D with the limit $I^\xi = I$.

(if part) Assume that $I \models \varphi_P$. The sequence $(I^n)_{n\leq\xi}$ is strictly increasing in precision, hence there is no $n \leq \xi$ such that $\varphi_P^{I^n} = \mathbf{f}$. As such, for every $n \leq \xi$, $P^{I^n} \neq \mathbf{f}$. Therefore, $P^I \neq \mathbf{f}$ and because I is two-valued, we can conclude $P^I = \mathbf{t}$.

(only if part) Assume that $I \models P$. Thus, for some $n < \xi$, $P^{I^n} = \mathbf{u}$ and $P^{I^{n+1}} = \mathbf{t}$. Hence, $\varphi_P^{I^n} = \mathbf{t}$. Because the sequence $(I^n)_{n\leq\xi}$ is strictly increasing in precision, we have $\varphi_P^I = \mathbf{t}$.

Lemma 3.4.4 (Soundness of the right definition rule) *Let D be a definition and P a defined atom of D. If $\models \Gamma \to \Delta, \varphi_P$, then $\models D, \Gamma \to \Delta, P$.*

Proof. Assume $\models \Gamma \to \Delta, \varphi_P$ but $\not\models D, \Gamma \to \Delta, P$. Then there exists a counter-model I for $D, \Gamma \to \Delta, P$ which satisfies D, $\bigwedge \Gamma$, $\neg \bigvee \Delta$ and $\neg P$. It follows from the first assumption that $I \models \varphi_P$, and hence, by Lemma 3.4.3, $I \models P$, a contradiction.

Lemma 3.4.5 (Soundness of the left definition rule) *Let D be a definition and U be a non-empty subset of $Def(D)$. If for every $P \in U$, it holds that $\models \neg U^\triangleright, \Gamma \to \Delta, \neg\varphi_P^\triangleright$, then for all $P \in U$, it holds that $\models P, D, \Gamma \to \Delta$.*

Proof. Assume $\models \neg U^\triangleright, \Gamma \to \Delta, \neg\varphi_P^\triangleright$ for every $P \in U$, but $\not\models P, D, \Gamma \to \Delta$ for some $P \in U$. Then there exists a model I of D, $\bigwedge \Gamma$ and $\neg \bigvee \Delta$ satisfying at least one $P \in U$. Furthermore, by Lemma 3.4.3, it holds that $I \models \varphi_P$. We select this P in the following way. Let $(I^n)_{n\leq\xi}$ be a terminal well-founded induction for D with limit $I^\xi = I$. Let n be the smallest $n \leq \xi$ such that for some $Q \in U$, $Q^{I^n} = \mathbf{u}$ and $Q^{I^{n+1}} = \mathbf{t}$. By selection of n, there is a unique $P \in U$ such that $P^{I^n} = \mathbf{u}$, $I^n \models \varphi_P$ and $P^{I^{n+1}} = \mathbf{t}$. Consider this P and φ_P.

On the one hand, it holds that $I \models \varphi_P$. On the other hand, consider the interpretation $I^\triangleright = I[U^\triangleright/\mathbf{f}]$. It is clear that $I^\triangleright$ satisfies $\neg U^\triangleright$, $\bigwedge \Gamma$ and $\neg \bigvee \Delta$. Hence, by the first assumption, it holds that $I^\triangleright \models \neg\varphi_P^\triangleright$. We will derive a contradiction from this.

Observe that by our choice of n, for each $Q \in U$, $Q^{I^n} = \mathbf{f}$ or $Q^{I^n} = \mathbf{u}$. Denote by $I^{n\rhd}$ the interpretation that assigns Q^{I^n} to $Q^\rhd$ for every $Q \in U$ and corresponds to I^n on all other atoms. There are two simple observations that can be made about $I^{n\rhd}$:

- $I^{n\rhd} \leq_p I^\rhd$: indeed, $I^n \leq_p I$ and for each $Q^\rhd \in U^\rhd$, $Q^{\rhd I^\rhd} = \mathbf{f} \geq_p Q^{\rhd I^{n\rhd}} = Q^{I^n} = \mathbf{f}$ or $\mathbf{u}$.
- $(\varphi_P^\rhd)^{I^{n\rhd}} = \varphi_P^{I^n} = \mathbf{t}$: obvious from the construction of $I^{n\rhd}$ and $\varphi_P^\rhd$.

Combining these results, we obtain $\mathbf{t} = (\varphi_P^\rhd)^{I^{n\rhd}} \leq_p (\varphi_P^\rhd)^{I^\rhd} = \mathbf{f}$. This is the desired contradiction.

Having the soundness of the left definition rule, we can explain the introduction of renaming formulas in the left definition rule. Our original idea for the left definition rule was of the form

$$\frac{\neg P_1, \ldots, \neg P_n, \Gamma \to \Delta, \neg\varphi_{P_1}; \ldots; \neg P_1, \ldots, \neg P_n, \Gamma \to \Delta, \neg\varphi_{P_n}}{P_i, D, \Gamma \to \Delta} \tag{3.1}$$

where $\{P_1, \ldots, P_n\} \subseteq Def(D)$ and P_i is an arbitrary defined atom in $\{P_1, \ldots, P_n\}$.

Intuitively, the above form of the left definition rule is exactly related to the second case of the well-founded induction and it is easier to be understood. However, such an inference rule is not sound. For an arbitrary definition D and any defined atom P of D, $D \to \neg P$ can be inferred by applying this rule. We illustrate this with the next example.

Example 3.4.6 *Consider the following definition:*

$$D = \left\{ \; P \leftarrow \top \; \right\}.$$

Let $\Gamma = \{P\}$ and Δ be an empty set. Since $\neg P, P \to \neg\top$, the following is a proof for $D \to \neg P$ by using the inference rule (3.1) *instead of left definition rule in it.*

$$\cfrac{\cfrac{\cfrac{\neg P, P \to \neg\top}{P, D, P \to}\ \textit{inference rule 3.1}}{P, D \to}\ \textit{left contraction}}{D \to \neg P}\ \textit{right } \neg$$

However, for the same definition D and empty sets Γ and Δ, we can have a proof for $D \to P$ as follows:

$$\cfrac{\to \top}{D \to P}\ \textit{right definition rule}$$

which derives a contradiction to $D \to \neg P$*. Hence, the inference rule* (3.1) *is not sound.*

From the viewpoint of semantics, since the left definition rule corresponds to the second case of the well-founded induction, we have to adopt the approach of renaming to represent that the defined atoms of U are unknown in I^n and false in I^{n+1}.

Lemma 3.4.7 *Let* D *be a definition,* I *a model of* D *and* U *a non-empty subset of* $Def(D)$*. If for every* $P \in U$*, it holds that* $\varphi_P^{I[U/\mathbf{f}]} = \mathbf{f}$*, then* $P^I = \mathbf{f}$ *for all* $P \in U$*.*

Proof. Assume that there exists a non-empty set T satisfying that (a) $T \subseteq U$, (b) $P^I = \mathbf{t}$ for each $P \in T$, and (c) $P^I = \mathbf{f}$ for each $P \in U \setminus T$. Let $(I^n)_{n \leq \xi}$ be a terminal well-founded induction for D with the limit $I^\xi = I$. Let n be the smallest $n \leq \xi$ such that for some $Q \in T$, $Q^{I^n} = \mathbf{u}$ and $Q^{I^{n+1}} = \mathbf{t}$. By selection of n, there is a unique $P \in T$ such that $P^{I^n} = \mathbf{u}$, $\varphi_P^{I^n} = \mathbf{t}$ and $P^{I^{n+1}} = \mathbf{t}$. Consider this P and φ_P.

Observe that by our choice of n, for each $Q \in T$, $Q^{I^n} = \mathbf{u}$. Hence, for each $Q \in T$, $Q^{I^n} \leq_p Q^{I[U/\mathbf{f}]} = \mathbf{f}$. Because $I^n \leq_p I$, for each $Q \in Def(D) \setminus T$, we have that $Q^{I[U/\mathbf{f}]} = Q^I \geq_p Q^{I^n}$. Combining these results, it is concluded that $I^n \leq_p I[U/\mathbf{f}]$. Therefore, we obtain that $\mathbf{t} = \varphi_P^{I^n} \leq_p \varphi_P^{I[U/\mathbf{f}]} = \mathbf{f}$, a contradiction. Hence, there is no $P \in U$ such that $P^I = \mathbf{t}$. It follows directly that $P^I = \mathbf{f}$ for all $P \in U$.

Recall that the notations used in the following lemma and the proof of the lemma have been given in the introduction of the non-total definition rule.

Lemma 3.4.8 *If* $\models V^\diamond, D^\diamond, \Gamma \to \Delta, \bigwedge \neg V^\triangleright$ *and* $\models \neg V^\diamond, D^\diamond, \Gamma \to \Delta, \bigwedge V^\triangleright$*, then* $\models D, \Gamma \to \Delta$*.*

Proof. Assume towards contradiction that

$$\models V^\diamond, D^\diamond, \Gamma \to \Delta, \bigwedge \neg V^\triangleright \text{ and } \models \neg V^\diamond, D^\diamond, \Gamma \to \Delta, \bigwedge V^\triangleright \text{ but } \not\models D, \Gamma \to \Delta. \tag{3.2}$$

Then there exists a Σ-interpretation I satisfying D, $\bigwedge \Gamma$ and $\neg \bigvee \Delta$. Consider the vocabulary $\Sigma^\diamond = \Sigma \cup V^\triangleright \cup V^\diamond$. I can be expanded into two $\Sigma^\diamond$-interpretations $I_{V^\diamond}$ and $I_{\neg V^\diamond}$ as follows:

$$I_{V^\diamond} = (I[V^\diamond/\mathbf{t}])^{D^\diamond} \quad \text{and} \quad I_{\neg V^\diamond} = (I[V^\diamond/\mathbf{f}])^{D^\diamond}.$$

Since $D^\diamond$ is a positive definition, hence total definition with open symbols $\Sigma \cup V^\diamond$, both interpretations are well-defined. Moreover they obviously satisfy:

$$I_{V^\diamond} \models \bigwedge V^\diamond \wedge D^\diamond \wedge \bigwedge \Gamma \wedge \neg \bigvee \Delta \quad \text{and} \quad I_{\neg V^\diamond} \models \bigwedge \neg V^\diamond \wedge D^\diamond \wedge \bigwedge \Gamma \wedge \neg \bigvee \Delta.$$

By (3.2), it follows that

$$I_{V^\diamond} \models \bigwedge \neg V^\triangleright \text{ and } I_{\neg V^\diamond} \models \bigwedge V^\triangleright. \tag{3.3}$$

Let $(I^n)_{n \leq \xi}$ be a terminal well-founded induction for D with limit $I^\xi = I$. There exists a least ordinal n such that $P^{I^n} = \mathbf{u}$ for every $P \in V$ and there exists at least one $P \in V$ with $P^{I^{n+1}} \neq \mathbf{u}$. We distinguish between the case where P is $\mathbf{t}$ in I^{n+1} and the case where $P \in U$ for some non-empty set $U \subseteq Def(D)$ such that all atoms of U are $\mathbf{f}$ in I^{n+1}. We will prove in the first case that $I_{V^\diamond} \models P^\triangleright$ and in the second case that $I_{\neg V^\diamond} \models \neg P^\triangleright$ for every $P^\triangleright \in V^\triangleright \cap U^\triangleright$. This contradicts (3.3).

- Assume that $P^{I^n} = \mathbf{u}$ and $P^{I^{n+1}} = \mathbf{t}$. Then for the rule $P \leftarrow \varphi_P \in D$, it holds that $\varphi_P^{I^n} = \mathbf{t}$. Consider the corresponding rule $P^\triangleright \leftarrow \varphi_P^\diamond \in D^\diamond$. If we can show that $I_{V^\diamond} \models \varphi_P^\diamond$, then Lemma 3.4.3 will yield that $I_{V^\diamond} \models P^\triangleright$ which is what we must prove here.

  Consider the $\Sigma^\diamond$-interpretation $I^{n\diamond}$ which extends I^n by interpreting each symbol $Q^\triangleright$ and $Q^\diamond$ as Q^{I^n}, i.e., as $\mathbf{u}$. Clearly, $(\varphi_P^\diamond)^{I^{n\diamond}} = \varphi_P^{I^n} = \mathbf{t}$, and it suffices to show that $I^{n\diamond} \leq_p I_{V^\diamond}$ to obtain that $I_{V^\diamond} \models \varphi_P^\diamond$. But this is straightforward since $I^{n\diamond}|_\Sigma = I^n \leq_p I = I_{V^\diamond}|_\Sigma$ and $(Q^\triangleright)^{I^{n\diamond}} = (Q^\diamond)^{I^{n\diamond}} = \mathbf{u}$ for each $Q \in V$. Hence, it is indeed the case that $I^{n\diamond} \leq_p I_{V^\diamond}$ which leads to the contradiction.

- For the other case, assume that $I^{n+1} = I^n[U/\mathbf{f}]$ where $P \in U$. For each $P \in U \cap V (\neq \emptyset)$ and its rule $P \leftarrow \varphi_P \in D$, it holds that $\varphi_P^{I^{n+1}} = \mathbf{f}$. We will use this to show that for each rule $P^\triangleright \leftarrow \varphi_P^\diamond \in D^\diamond$ with $P^\triangleright \in U^\triangleright \cap V^\triangleright$, $\varphi_P^\diamond$ is false in the interpretation $I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]$. Then, since $I_{\neg V^\diamond}$ satisfies $D^\diamond$, we can apply Lemma 3.4.7 to obtain that each $P^\triangleright \in U^\triangleright \cap V^\triangleright$ is false in $I_{\neg V^\diamond}$. This produces the contradiction with (3.3).

  The key point is therefore to show that all these renamed rule bodies $\varphi_P^\diamond$ are false in the interpretation $I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]$. To prove this, we use the same technique as in the previous case, namely we construct an interpretation which is less precise than $I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]$ and which falsifies all the concerned rule bodies. We choose this interpretation as the $\Sigma^\diamond$-interpretation $I^\diamond$ which extends I^{n+1} by interpreting each symbol $Q^\triangleright$ and $Q^\diamond$ as $Q^{I^{n+1}}$, i.e., as $\mathbf{f}$ if $Q \in U \cap V$ and as $\mathbf{u}$ if $Q \in V \setminus U$. Notice that for all formulas ψ over Σ, it holds that $\psi^{I^{n+1}} = (\psi^\diamond)^{I^\diamond}$.

Let us verify that $I^\diamond \leq_p I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]$. We have $I^\diamond|_\Sigma = I^{n+1} \leq_p I = I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]|_\Sigma$. The interpretation $I^\diamond$ interprets all symbols $Q^\diamond$ as $\mathbf{u}$ or $\mathbf{f}$ whereas $I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]$ interprets them as $\mathbf{f}$, just like $I_{\neg V^\diamond}$. Symbols of $U^\triangleright \cap V^\triangleright$ are interpreted as $\mathbf{f}$ in both interpretations, and finally, the remaining symbols of $V^\triangleright \setminus U^\triangleright$ are interpreted as $\mathbf{u}$ in $I^\diamond$ which is certainly less precise than in the other interpretation.

It follows that for every rule $P^\triangleright \leftarrow \varphi_P^\diamond \in D^\diamond$ with $P \in V \cap U$, $\mathbf{f} = \varphi_P^{I^{n+1}} = (\varphi_P^\diamond)^{I^\diamond} \leq_p (\varphi_P^\diamond)^{I_{\neg V^\diamond}[U^\triangleright \cap V^\triangleright/\mathbf{f}]} = \mathbf{f}$. As explained before, this leads to the desired contradiction.

The renamings in the non-total definition rule may seem cumbersome. However, our original form of the non-total definition rule, which does not contain any renaming in it, is not sound. Even one renaming does not seem sufficient to obtain a sound non-total definition rule. Thus, we adopt two renamings in the current version of the non-total definition rule.

Lemma 3.4.9 (Soundness of the definition introduction rule) *Let D be a total definition. If $\models D', \Gamma \to \Delta, P' \equiv P$ for every $P \in Def(D)$, then $\models \Gamma \to \Delta, D$.*

Proof. Assume $\models D', \Gamma \to \Delta, P' \equiv P$ for every $P \in Def(D)$ but $\not\models \Gamma \to \Delta, D$. Then there exists a two-valued Σ-interpretation I such that $I \models \bigwedge \Gamma$ but $I \not\models \bigvee \Delta$, $I \not\models D$. Denote by J the two-valued well-founded model of D extending $I|_{Open(D)}$. Because $I \not\models D$, there exists a defined atom Q of D such that $Q^I \neq Q^J$. Since D is a total definition and D' is obtained by replacing each occurrence of each defined atom P in D by P', D' is a total definition. Thus, there exists a two-valued Σ'-interpretation I' such that I' is the well-founded model of D' extending I. Notice that for every $P \in Def(D)$, $P^{I'} = P^I$. Because neither Γ nor Δ contains an occurrence of an atom P', it holds that $I' \models \bigwedge \Gamma$ and $I' \not\models \bigvee \Delta$. Therefore, by the first assumption, it is obtained that $I' \models P' \equiv P$ for every $P \in Def(D)$. Also, because D' is obtained by renaming all defined atoms and none of the open atoms, it holds that $P^J = (P')^{I'}$ for every $P \in Def(D)$. Hence, $Q^I = Q^{I'} = (Q')^{I'} = Q^J$, a contradiction. Therefore, $\models \Gamma \to \Delta, D$.

The definition introduction rule is not sound if the inferred definition D is not total. We illustrate it with an example.

Example 3.4.10 *Consider the definition as follows:*

$$D = \left\{ \; P \leftarrow \neg P \; \right\}.$$

Let Γ and Δ be empty sets. It is obvious that $D' = \left\{ \; P' \leftarrow \neg P' \; \right\}$ is not total. Thus, $\models D' \to P' \equiv P$ but $\not\models\!\!\to D$, which shows that the definition introduction rule is not sound when the inferred definition D is non-total.

It follows that all inference rules in **LPC(ID)** except the definition introduction rule are sound with respect to both total and non-total definitions. By induction on the number of inference rules in a proof of a sequent, we obtain the soundness theorem of **LPC(ID)**.

Theorem 3.4.11 (Soundness of LPC(ID)) *Let Γ be an arbitrary set of PC(ID) formulas and Δ an arbitrary set of PC formulas. If $\Gamma \rightarrow \Delta$ is provable in* **LPC(ID)**, *then $\models \Gamma \rightarrow \Delta$. Let Γ' and Δ' be sets of PC(ID) formulas such that all definitions occurring in Γ' and Δ' are total. If $\Gamma' \rightarrow \Delta'$ is provable in* **LPC(ID)**, *then $\models \Gamma' \rightarrow \Delta'$.*

3.5 Completeness of LPC(ID)

We now present a proof of the *completeness* property of **LPC(ID)**, namely that the converse of the soundness theorem holds: if $\Gamma \rightarrow \Delta$ is valid, then there exists an **LPC(ID)** proof of $\Gamma \rightarrow \Delta$. **LPC(ID)** is not complete in general. Intuitively, this is because the only inference rules that allows to introduce a positive occurrence of a definition in the succedent of a sequent are the axioms, the weakening rules and the definition introduction rule. As shown in the above section, the definition introduction rule is not sound with respect to non-total definitions. Also, no other inference rule allows a proof to derive a non-total definition from some propositional formulas. Therefore, one cannot synthesize non-total definitions with **LPC(ID)**, i.e., not all valid sequents of the form $\Gamma \rightarrow D$, where D is a non-total definition, can be proved in this deductive system.

We will however prove the completeness for a restricted class of sequents, namely the sequents $\Gamma \rightarrow \Delta$ such that every definition occurring negatively in Γ or positively in Δ must be total. Our completeness proof employs an extension of the direct style of completeness proof for calculi for propositional logic as given in e.g., [172, 173]. The main difficulty in the completeness proof for **LPC(ID)** is to handle the definitions appearing in the sequents.

For the sake of completeness, we first spell out the details of the proof of the completeness for the calculi for propositional logic. We follow a standard technique [173], namely we construct the so-called *reduction tree* for a sequent $\Gamma \rightarrow \Delta$, where Γ and Δ are sets of PC formulas. We introduce some terminology as follows.

Definition 3.5.1 *A* reduction tree *for a sequent $S = \Gamma \rightarrow \Delta$, is a tree T_S of sequents with root S. Moreover, T_S is constructed by applying one of the following reductions on each non-leaf sequent $\Pi \rightarrow \Sigma$.*

- *(left $\neg$ reduction)* Π *contains a sequent formula* $\neg A$*, then write down* $\Pi \setminus \{\neg A\} \to \Sigma, A$ *as the unique child of* $\Pi \to \Sigma$*.*
- *(right $\neg$ reduction)* Σ *contains a sequent formula* $\neg A$*, then write down* $A, \Pi \to \Sigma \setminus \{\neg A\}$ *as the unique child of* $\Pi \to \Sigma$*.*
- *(left $\wedge$ reduction)* Π *contains a sequent formula* $A \wedge B$*, then write down* $A, B, \Pi \setminus \{A \wedge B\} \to \Sigma$ *as the unique child of* $\Pi \to \Sigma$*.*
- *(right $\wedge$ reduction)* Σ *contains a sequent formula* $A \wedge B$*, then write down* $\Pi \to \Sigma \setminus \{A \wedge B\}, A$ *and* $\Pi \to \Sigma \setminus \{A \wedge B\}, B$ *as two children of* $\Pi \to \Sigma$*.*
- *(left $\vee$ reduction)* Π *contains a sequent formula* $A \vee B$*, then write down* $A, \Pi \setminus \{A \vee B\} \to \Sigma$ *and* $B, \Pi \setminus \{A \vee B\} \to \Sigma$ *as two children of* $\Pi \to \Sigma$*.*
- *(right $\vee$ reduction)* Σ *contains a sequent formula* $A \vee B$*, then write down* $\Pi \to \Sigma \setminus \{A \vee B\}, A, B$ *as the unique child of* $\Pi \to \Sigma$*.*

In addition, each leaf of T_S *is either an axiom, or none of the above reductions is possible.*

Observe that each reduction respectively corresponds to a logical inference rule. Each leaf node of a reduction tree is either an axiom or a sequent of the form $\Gamma \to \Delta$ where Γ and Δ are sets of atoms with $\Gamma \cap \Delta = \emptyset$.

Definition 3.5.2 *An inference rule* preserves a counter-model *if for each instance of the inference rule, a counter-model for one of the premises of the instance is the same as a counter-model for the conclusion of the instance.*

The following property can easily be verified.

Proposition 3.5.3 *All the logical inference rules preserves counter-models.*

Then we obtain the property of reduction trees as follows.

Proposition 3.5.4 *For each sequent* $S = \Gamma \to \Delta$*, (a) there exists a reduction tree* T_S*, (b) if all leaf nodes of a reduction tree* T_S *are provable, then the root sequent is provable, and (c) , If* T_S *has a leaf node for which a counter-model exists, then this counter-model is also a counter-model for the root.*

Proof. Clearly, a reduction tree exists because it can be constructed by a non-deterministic reduction process. Because each reduction in a reduction tree corresponds to a logical inference rule, by using the corresponding inference rule,

it is easy to prove that if the children of a node in a reduction tree are provable, then the node itself is provable. Therefore, the root sequent is provable if all leaf nodes of the reduction tree are provable.

A counter-model for a leaf is a counter-model for the root because all the logical inference rules preserve counter-models by Proposition 3.5.3 and each non-leaf node can be proved from its children using only those inference rules.

We are now ready to prove the completeness theorem of the calculi for propositional logic.

Theorem 3.5.5 (Completeness for propositional logic) *Let* Γ *and* Δ *be sets of PC formulas. If* $\models \Gamma \to \Delta$*, then* $\Gamma \to \Delta$ *is provable.*

Proof. First, a reduction tree is constructed from the root $\Gamma \to \Delta$. Every leaf node of the reduction tree must be an axiom or a sequent of the form $\Pi \to \Sigma$, where Π and Σ are sets of propositional atoms satisfying that Π and Σ have no atom in common. Suppose that there is a leaf node $\Pi \to \Sigma$ of the reduction tree such that $\Pi \to \Sigma$ is not an axiom and Π, Σ are sets of propositional atoms with $\Pi \cap \Sigma = \emptyset$. We construct an interpretation I such that $I(P) = \mathbf{t}$ for each $P \in \Pi$, $I(P) = \mathbf{f}$ for each $P \in \Sigma$ and $I(P) = \mathbf{t}$ for every other atom. It is trivial to verify that I is a counter-model for the leaf node $\Pi \to \Sigma$. Thus by (c) of Proposition 3.5.4, it is known that I is a counter-model for the root $\Gamma \to \Delta$, which is a contradiction to the fact that $\Gamma \to \Delta$ is valid. It follows that each leaf node of the reduction tree is an axiom, and thus, each leaf node is provable. Hence, by (b) of Proposition 3.5.4, $\Gamma \to \Delta$ is provable, as desired.

We are now in a position to prove the completeness of **LPC(ID)**. First, we focus on the completeness of sequents of the form $D, \Gamma \to \Delta$, where Γ and Δ are sets of PC formulas and D is a definition. Notice that the definition D appearing in the sequent may be non-total.

Lemma 3.5.6 *Let* D *be a definition and* Γ *a set of open literals of* D *such that for every* $Q \in Open(D)$ *either* $Q \in \Gamma$ *or* $\neg Q \in \Gamma$*. Let* I_O *be the unique two-valued* $Open(D)$*-interpretation such that* $I_O \models \bigwedge \Gamma$ *and* I *the well-founded model of* D *extending* I_O*. If* L *is a defined literal of* D *such that* $L^I = \mathbf{t}$*, then* $D, \Gamma \to L$ *is provable in* **LPC(ID)**.

Proof. Let $(I^n)_{n \leq \xi}$ be a terminal well-founded induction for D extending I_O with the limit $I^\xi = I$. Denote by Δ^n the set of all defined literals L such that $L^{I^n} = \mathbf{t}$ in arbitrary order. We prove that $\Delta^n, D, \Gamma \to L$ is provable in **LPC(ID)** for all $L \in \Delta^{n+1} \setminus \Delta^n$. For each $L \in \Delta^{n+1} \setminus \Delta^n$, $L^{I^n} = \mathbf{u}$ and $L^{I^{n+1}} = \mathbf{t}$. We distinguish

between the case where $\Delta^{n+1} \setminus \Delta^n$ contains one positive literal and the case where it contains a set of negative literals.

- Assume that $\Delta^{n+1} \setminus \Delta^n$ consists of one defined atom P. For every two-valued Σ-interpretation J such that J is a model of $\bigwedge \Gamma$ and $\bigwedge \Delta^n$, $I^n \leq_p J$. Indeed, $L^{I^n} = L^J = \mathbf{t}$ for every $L \in \Gamma$, $L^{I^n} = L^J = \mathbf{t}$ for every $L \in \Delta^n$ and for every other atom $Q \in \Sigma$, $Q^{I^n} = \mathbf{u} \leq_p Q^J$. $P^{I^{n+1}} = \mathbf{t}$, hence $\varphi_P^{I^n} = \mathbf{t}$. It follows that $\varphi_P^J = \mathbf{t}$. Thus, $\models \Delta^n, \Gamma \to \varphi_P$. Therefore, by the completeness of the propositional part of **LPC(ID)**, the sequent $\Delta^n, \Gamma \to \varphi_P$ is provable in **LPC(ID)**. Hence, by the right definition rule, $\Delta^n, D, \Gamma \to P$ is provable in **LPC(ID)**.
- For the other case, assume that $\Delta^{n+1} \setminus \Delta^n$ is a set of negative literals. Denote the set $\{P \mid \neg P \in \Delta^{n+1} \setminus \Delta^n\}$ by U. Recall that $I^{n+1} = I^n[U/\mathbf{f}]$. $P^{I^{n+1}} = \mathbf{f}$ for each $P \in U$, hence $\varphi_P^{I^{n+1}} = \mathbf{f}$. Consider the interpretation $I^{n+1^\triangleright} = I^n[U^\triangleright/\mathbf{f}]$. There are two simple observations that can be made about $I^{n+1^\triangleright}$ and each $\varphi_P^\triangleright$:
  - $I^{n+1^\triangleright} \leq_p J'$ for every two-valued $\Sigma \cup U^\triangleright$-interpretation J' such that J' satisfies $\bigwedge \Gamma$, $\bigwedge \Delta^n$ and $\bigwedge \neg U^\triangleright$: indeed, $L^{I^{n+1^\triangleright}} = L^{J'} = \mathbf{t}$ for every $L \in \Gamma$, $L^{I^{n+1^\triangleright}} = L^{J'} = \mathbf{t}$ for every $L \in \Delta^n$, $(P^\triangleright)^{I^{n+1^\triangleright}} = P^{\triangleright J'} = \mathbf{f}$ for every $P^\triangleright \in U^\triangleright$ and $Q^{I^{n+1^\triangleright}} = \mathbf{u} \leq_p Q^{J'}$ for every other atom $Q \in \Sigma \cup U^\triangleright$.
  - $(\varphi_P^\triangleright)^{I^{n+1^\triangleright}} = \varphi_P^{I^{n+1}} = \mathbf{f}$: obvious from the construction of $I^{n+1^\triangleright}$ and $\varphi_P^\triangleright$.

  Combining these results, we obtain $(\varphi_P^\triangleright)^{J'} = \mathbf{f}$ for every two-valued interpretation J' satisfying $\bigwedge \Gamma$, $\bigwedge \Delta^n$ and $\bigwedge \neg U^\triangleright$. It follows that $\models \neg U^\triangleright, \Delta^n, \Gamma \to \neg \varphi_P^\triangleright$ for every $P \in U$. By the completeness of the propositional part of **LPC(ID)**, the left definition rule and the right $\neg$ rule the sequent $\Delta^n, D, \Gamma \to \neg P$ is provable in **LPC(ID)** for every $P \in U$.

Since $(I^n)_{n \leq \xi}$ is a terminal well-founded induction for D with the limit $I = I^\xi$, it is obvious that the set of defined literals L for which $L^I = \mathbf{t}$ is exactly the set of all defined literals in Δ^ξ. Thus, by using the cut rule, it is easy to show by induction on n that if L is a defined literal of D such that $L^I = \mathbf{t}$, the sequent $D, \Gamma \to L$ is provable in **LPC(ID)**.

Notice that in the above lemma, we do not require the totality of the definition. So the definition D can be non-total and the well-founded model of D may be a three-valued interpretation.

Lemma 3.5.7 *Let D be a total definition and let Γ be a set of open literals of D, such that for every atom $Q \in Open(D)$ either $Q \in \Gamma$ or $\neg Q \in \Gamma$. Let L be a defined literal of D. If $\models D, \Gamma \to L$, then $D, \Gamma \to L$ is provable in* **LPC(ID)**.

Proof. Assume that $\models D, \Gamma \rightarrow L$. Let I_O be the unique two-valued $Open(D)$-interpretation such that $I_O \models \bigwedge \Gamma$. Because D is total, I_O can be extended to a two-valued well-founded model I of D such that $I \models \bigwedge \Gamma$ and $I \models D$. Then since $\models D, \Gamma \rightarrow L$, it holds that $L^I = \mathbf{t}$. Thus, by Lemma 3.5.6, $D, \Gamma \rightarrow L$ is provable in **LPC(ID)**.

Lemma 3.5.8 *Let D be a total definition and Γ an arbitrary consistent set of literals. If L is a defined literal of D such that $\models D, \Gamma \rightarrow L$, then $D, \Gamma \rightarrow L$ is provable in* **LPC(ID)**.

Proof. Let Γ' be an arbitrary extension of Γ such that for every open atom Q of D, either $Q \in \Gamma'$ or $\neg Q \in \Gamma'$. First, we want to show that $D, \Gamma' \rightarrow L$ is provable in **LPC(ID)**. It holds that $\models D, \Gamma' \rightarrow L$ because $\models D, \Gamma \rightarrow L$. Consider the set Γ'' of all open literals of D in Γ'. If $\models D, \Gamma'' \rightarrow L$, then by the previous lemma, $D, \Gamma'' \rightarrow L$ is provable in **LPC(ID)**, and by the left weakening rule, so is $D, \Gamma' \rightarrow L$. If $\not\models D, \Gamma'' \rightarrow L$, then by totality of D, $\models D, \Gamma'' \rightarrow \neg L$ and hence, $\models D, \Gamma' \rightarrow \neg L$. This means that $D \wedge \bigwedge \Gamma'$ is unsatisfiable, which implies that for some defined literal L' in Γ', $\models D, \Gamma'' \rightarrow \neg L'$. By the previous lemma and the left weakening rule, $D, \Gamma' \rightarrow \neg L'$ is provable in **LPC(ID)**. It is obvious that $D, \Gamma' \rightarrow L'$ is an axiom because L' is a literal in Γ'. Then we can use the left $\neg$ rule, the cut rule and the right weakening rule to show that $D, \Gamma' \rightarrow L$ is provable in **LPC(ID)**.

Given that the sequents $D, \Gamma' \rightarrow L$ are provable in **LPC(ID)** for all extensions Γ' of Γ, by using the right $\neg$ rule and the cut rule on all $D, \Gamma' \rightarrow L$, an **LPC(ID)**-proof for $D, \Gamma \rightarrow L$ can be constructed.

Lemma 3.5.9 *Let D be a definition and Γ a set of open literals of D, such that for every atom $Q \in Open(D)$ either $Q \in \Gamma$ or $\neg Q \in \Gamma$. If $\models D, \Gamma \rightarrow \bot$, then $D, \Gamma \rightarrow \bot$ is provable in* **LPC(ID)**.

Proof. Let I_O be the unique two-valued $Open(D)$-interpretation such that $I_O \models \bigwedge \Gamma$ and $(I^n)_{n \leq \xi}$ a terminal well-founded induction for D extending I_O with limit $I^\xi = I$. Because $\models D, \Gamma \rightarrow \bot$, there is no two-valued well-founded model for D extending I_O. Hence I is a three-valued Σ-interpretation. Denote by E the set of all defined atoms of D which are not unknown in I and V the set $Def(D) \setminus E$. For each $P \in E$, we define a literal L_P as follows:

$$L_P = \begin{cases} P & \text{if } P^I = \mathbf{t} \\ \neg P & \text{if } P^I = \mathbf{f} \end{cases}.$$

Denote by K the set $\{L_P \mid P \in E\}$ of literals. We first want to show that

$$\models D^\diamond, V^\diamond, K, \Gamma \rightarrow \bigwedge \neg V^\triangleright \text{ and } \models D^\diamond, \neg V^\diamond, K, \Gamma \rightarrow \bigwedge V^\triangleright. \tag{3.4}$$

Consider the vocabulary $\Sigma^\diamond = \Sigma \cup V^\triangleright \cup V^\diamond$. I can be expanded into two $\Sigma^\diamond$-interpretations $I_{V^\diamond}$ and $I_{\neg V^\diamond}$ as follows:

$$I_{V^\diamond} = (I[V^\diamond/\mathbf{t}])^{D^\diamond} \quad \text{and} \quad I_{\neg V^\diamond} = (I[V^\diamond/\mathbf{f}])^{D^\diamond}.$$

Since $D^\diamond$ is a positive definition, hence total definition with open symbols $\Sigma \cup V^\diamond$, both interpretations are well-defined. Moreover it is obvious that $I_{V^\diamond}$, respectively $I_{\neg V^\diamond}$, is the only interpretation satisfying:

$$I_{V^\diamond} \models D^\diamond \wedge \bigwedge V^\diamond \wedge \bigwedge K \wedge \bigwedge \Gamma, \text{ respectively } I_{\neg V^\diamond} \models D^\diamond \wedge \bigwedge \neg V^\diamond \wedge \bigwedge K \wedge \bigwedge \Gamma.$$

In order to prove (3.4), it suffices to show that

$$I_{V^\diamond} \models \bigwedge \neg V^\triangleright \text{ and } I_{\neg V^\diamond} \models \bigwedge V^\triangleright. \tag{3.5}$$

- We want to prove that $I_{V^\diamond} \models \bigwedge \neg V^\triangleright$. For any $P \in V$ with its rule $P \leftarrow \varphi_P \in D$, $P^\triangleright \leftarrow \varphi_P^\diamond$ is the corresponding rule for $P^\triangleright$ in $D^\diamond$. If we can show that $I_{V^\diamond}[V^\triangleright/\mathbf{f}] \models \neg\varphi_P^\diamond$ for each $P^\triangleright \in V^\triangleright$ with its rule $P^\triangleright \leftarrow \varphi_P^\diamond$, then since $I_{V^\diamond}$ satisfies $D^\diamond$, we can apply Lemma 3.4.7 to obtain that each $P^\triangleright \in V^\triangleright$ is false in $I_{V^\diamond}$, which is what we must prove here.

  Consider the $\Sigma^\diamond$-interpretation $I^\diamond$ which extends I by interpreting each symbol $Q^\triangleright$ and $Q^\diamond$ as Q^I for each $Q \in V$, i.e., as $\mathbf{u}$. Clearly, for every $P \in V$ with its rule $P \leftarrow \varphi_P \in D$, $(\varphi_P^\diamond)^{I^\diamond} = \varphi_P^I = \mathbf{u}$, and since $(\varphi_P^\diamond)^{I_{V^\diamond}[V^\triangleright/\mathbf{f}]} \neq \mathbf{u}$, it is sufficient to show that $(\varphi_P^\diamond)^{I_{V^\diamond}[V^\triangleright/\mathbf{f}]} \leq (\varphi_P^\diamond)^{I^\diamond}$ to obtain that $I_{V^\diamond}[V^\triangleright/\mathbf{f}] \models \neg\varphi_P^\diamond$ for every $P^\triangleright \in V^\triangleright$ with its rule $P^\triangleright \leftarrow \varphi_P^\diamond \in D^\diamond$. This can be verified by the following observations.

  - $I_{V^\diamond}[V^\triangleright/\mathbf{f}]|_\Sigma = I^\diamond|_\Sigma$.
  - For every $Q \in V$, every occurrence of $Q^\triangleright$ in $\varphi_P^\diamond$ is positive and $(Q^\triangleright)^{I_{V^\diamond}[V^\triangleright/\mathbf{f}]} = \mathbf{f} \leq (Q^\triangleright)^{I^\diamond} = \mathbf{u}$.
  - For every $Q \in V$, every occurrence of $Q^\diamond$ in $\varphi_P^\diamond$ is negative and $(Q^\diamond)^{I_{V^\diamond}[V^\triangleright/\mathbf{f}]} = \mathbf{t} \geq (Q^\diamond)^{I^\diamond} = \mathbf{u}$

  Hence, it is indeed the case that $(\varphi_P^\diamond)^{I_{V^\diamond}[V^\triangleright/\mathbf{f}]} \leq (\varphi_P^\diamond)^{I^\diamond}$, as desired.

- We want to prove that $I_{\neg V^\diamond} \models \bigwedge V^\triangleright$. Assume towards contradiction that there exists a non-empty set $F^\triangleright \subseteq V^\triangleright$ such that $I_{\neg V^\diamond} \models \bigwedge \neg F^\triangleright$ and for the set $T^\triangleright = V^\triangleright \setminus F^\triangleright$, $I_{\neg V^\diamond} \models \bigwedge T^\triangleright$. Consider the Σ-interpretation $I^1 = I[F/\mathbf{f}]$. If we can show that $\varphi_P^{I^1} = \mathbf{f}$ for every $P \in F$ with its rule $P \leftarrow \varphi_P \in D$, then since for each $P \in F$ and its rule $P \leftarrow \varphi_P \in D$, $P^I = \mathbf{u}$ and $\varphi_P^{I^1} = \mathbf{f}$, I can be extended to I^1 in the well-founded induction $(I^n)_{n \leq \xi}$ for D. This produces the contradiction to that I is the limit of $(I^n)_{n \leq \xi}$. To prove that $\varphi_P^{I^1} = \mathbf{f}$ for every $P \in F$ with the rule $P \leftarrow \varphi_P \in D$, we first choose a $\Sigma^\diamond$-interpretation $I^\diamond$ which extends I^1 by interpreting each symbol $Q^\triangleright$ and $Q^\diamond$ as

Q^{I^1}, i.e., as $\mathbf{f}$ if $Q \in F$ and as $\mathbf{u}$ if $Q \in T$. Notice that for all formulas ψ over Σ, it holds that $\psi^{I^1} = (\psi^\diamond)^{I^\diamond}$. Thus, it is sufficient to show that $(\varphi_P^\diamond)^{I^\diamond} = \mathbf{f}$ for every $P^\triangleright \in F^\triangleright$ with the rule $P^\triangleright \leftarrow \varphi_P^\diamond \in D^\diamond$. Since $I_{\neg V^\diamond} \models \neg P^\triangleright$ for each $P^\triangleright \in F^\triangleright$ and $I_{\neg V^\diamond}$ is a model of $D^\diamond$, by Lemma 3.4.3, we have that $(\varphi_P^\diamond)^{I_{\neg V^\diamond}} = \mathbf{f}$ for every $P^\triangleright \in F^\triangleright$ with the rule $P^\triangleright \leftarrow \varphi_P^\diamond \in D^\diamond$. If we can have that $(\varphi_P^\diamond)^{I^\diamond} \leq (\varphi_P^\diamond)^{I_{\neg V^\diamond}} = \mathbf{f}$, it holds that $(\varphi_P^\diamond)^{I^\diamond} = \mathbf{f}$, which is exactly what we need.

We can verify that $(\varphi_P^\diamond)^{I^\diamond} \leq (\varphi_P^\diamond)^{I_{\neg V^\diamond}}$ by the following facts.

- $I_{\neg V^\diamond}|_\Sigma = I^\diamond|_\Sigma$.
- Every occurrence of $Q^\triangleright$ in $\varphi_P^\diamond$ is positive and $(Q^\triangleright)^{I_{\neg V^\diamond}} = (Q^\triangleright)^{I^\diamond} = \mathbf{f}$ for each $Q^\triangleright \in F^\triangleright$ while $(Q^\triangleright)^{I^\diamond} = \mathbf{u} \leq (Q^\triangleright)^{I_{\neg V^\diamond}} = \mathbf{t}$ for each $Q^\triangleright \in V^\triangleright \setminus F^\triangleright$.
- Every occurrence of $Q^\diamond$ in $\varphi_P^\diamond$ is negative and $(Q^\diamond)^{I_{\neg V^\diamond}} = (Q^\diamond)^{I^\diamond} = \mathbf{f}$ for each $Q^\diamond \in F^\diamond$ while $(Q^\diamond)^{I^\diamond} = \mathbf{u} \geq (Q^\diamond)^{I_{\neg V^\diamond}} = \mathbf{f}$ for each $Q^\diamond \in V^\diamond \setminus F^\diamond$.

Hence, it is the case that $(\varphi_P^\diamond)^{I^\diamond} \leq (\varphi_P^\diamond)^{I_{\neg V^\diamond}} = \mathbf{f}$, as desired.

Therefore, it is obtained that $\models D^\diamond, V^\diamond, K, \Gamma \to \bigwedge \neg V^\triangleright$ and $\models D^\diamond, \neg V^\diamond, K, \Gamma \to \bigwedge V^\triangleright$. $D^\diamond$ is a total definition, hence by using Lemma 3.5.8 and the right $\wedge$ rule, both $V^\diamond, D^\diamond, K, \Gamma \to \bigwedge \neg V^\triangleright$ and $\neg V^\diamond, D^\diamond, K, \Gamma \to \bigwedge V^\triangleright$ are provable in **LPC(ID)**. It follows from the non-total definition rule that $K, D, \Gamma \to \bot$ is provable in **LPC(ID)**. Since I is a well-founded model of D extending I_O and $L^I = \mathbf{t}$ for each $L \in K$, using Lemma 3.5.6, it holds that for each $L \in K$, $D, \Gamma \to L$ is provable in **LPC(ID)**. Consequently, by the multiple use of the cut rule on $K, D, \Gamma \to \bot$ and $D, \Gamma \to L$ for each $L \in K$, $D, \Gamma \to \bot$ is provable in **LPC(ID)**.

Lemma 3.5.10 *Let D be a definition and Γ a set of open literals of D such that for every atom $Q \in Open(D)$, either $Q \in \Gamma$ or $\neg Q \in \Gamma$. Let L be a defined literal of D. If $\models D, \Gamma \to L$, then $D, \Gamma \to L$ is provable in* **LPC(ID)**.

Proof. Assume $\models D, \Gamma \to L$. Let I_O be the unique two-valued $Open(D)$-interpretation such that $I_O \models \bigwedge \Gamma$. If $\not\models D, \Gamma \to \bot$, then I_O can be extended to the two-valued well-founded model I of D such that $I \models \bigwedge \Gamma$ and $I \models D$. Since $\models D, \Gamma \to L$, it holds that $I \models L$. Thus, by Lemma 3.5.6, $D, \Gamma \to L$ is provable in **LPC(ID)**. If $\models D, \Gamma \to \bot$, then by Lemma 3.5.9, $D, \Gamma \to \bot$ is provable in **LPC(ID)**. Hence, by the right weakening rule, $D, \Gamma \to L$ is provable in **LPC(ID)**.

Lemma 3.5.11 *Let D be a definition and Γ an arbitrary consistent set of literals. If $\models D, \Gamma \to \bot$, then $D, \Gamma \to \bot$ is provable in* **LPC(ID)**.

To prove this, we use the same technique as in the proof of Lemma 3.5.8. We omit the details of the proof here.

Lemma 3.5.12 *Let D be a definition, Γ an arbitrary consistent set of literals and L a defined literal of D. If $\models D, \Gamma \rightarrow L$, then $D, \Gamma \rightarrow L$ is provable in* **LPC(ID)**.

Proof. If $\Gamma \cup \{\neg L\}$ is an inconsistent set of literals, we have that $L \in \Gamma$ and hence, $D, \Gamma \rightarrow L$ is an axiom and thus, $D, \Gamma \rightarrow L$ is provable in **LPC(ID)**. If $\Gamma \cup \{\neg L\}$ is consistent, because $\models D, \Gamma, \neg L \rightarrow \bot$, by the previous lemma, it is obtained that $D, \Gamma, \neg L \rightarrow \bot$ is provable in **LPC(ID)**. Then by the $\neg$ rules and the cut rule, we can conclude that $D, \Gamma \rightarrow L$ is provable in **LPC(ID)**.

The remainder of the completeness proof for the class of sequents, namely the sequents $\Gamma, D \rightarrow \Delta$ where Γ and Δ are sets of PC formulas and D is a definition, will follow the standard approach of constructing a reduction tree for a sequent $\Gamma \rightarrow \Delta$, which is used in the completeness proof of propositional logic. We need to add another reduction, which is called *definition introduction reduction* and given as follows, in Definition 3.5.1 to obtain the reduction tree for a sequent containing definitions.

– (definition introduction reduction) Σ contains a sequent formula D, which is a total definition with $Def(D) = \{P_1, \ldots, P_n\}$, then write down $D', \Pi \rightarrow \Sigma \setminus \{D\}, P'_i \equiv P_i$ for each $i \in \{1, \ldots, n\}$ as n children of $\Pi \rightarrow \Sigma$, where D' is the definition over the new vocabulary $\Sigma' = \Sigma \cup \{P' \mid P \in Def(D)\}$ obtained by replacing each occurrence of each defined symbol P in D by P'.

Observe that the definition introduction reduction corresponds to the definition introduction rule. Each leaf node of a reduction tree is either an axiom or a sequent of the form $D_1, \ldots, D_n, \Gamma \rightarrow \Delta$ where Γ and Δ are sets of atoms with $\Gamma \cap \Delta = \emptyset$ and $D_1, \ldots, D_n$ are definitions.

Lemma 3.5.13 *The definition introduction rule preserves counter-models.*

Proof. Let D be a total definition. Then D' is a total definition because of its construction. Assume that I is a counter-model of $D', \Gamma \rightarrow \Delta, P' \equiv P$ for some $P \in Def(D)$, but I is not a counter-model of $\Gamma \rightarrow \Delta, D$. Since D and D' are total, I is a two-valued interpretation satisfying D', $\bigwedge \Gamma$, $\neg \bigvee \Delta$ and $\neg(P' \equiv P)$. Because I is not a counter-model for $\Gamma \rightarrow \Delta, D$, it holds that $I \models D$. Obviously from the construction of D' and the fact that I satisfies both D and D', we conclude that $I \models P' \equiv P$ for every $P \in Def(D)$, a contradiction.

Then we obtain the property of reduction trees as follows, which is quite similar to Proposition 3.5.4.

Proposition 3.5.14 *For each sequent $S = \Gamma \to \Delta$, (a) there exists a reduction tree T_S, (b) if all leaf nodes of a reduction tree T_S are provable in* **LPC(ID)**, *then the root sequent is provable in* **LPC(ID)**, *and, (c) if T_S has a leaf node for which a counter-model exists, then this counter-model is also a counter-model for the root.*

Proof. Clearly, a reduction tree exists because it can be constructed by a non-deterministic reduction process. Because each reduction in a reduction tree corresponds to either the definition introduction rule or a logical inference rule, by using the corresponding inference rule, it is easy to prove that if the children of a node in a reduction tree are provable in **LPC(ID)**, then the node itself is provable in **LPC(ID)**. Therefore, the root sequent is provable in **LPC(ID)** if all leaf nodes of the reduction tree are provable in **LPC(ID)**.

A counter-model for a leaf is a counter-model for the root because all the logical inference rules and the definition introduction rule preserve counter-models by Proposition 3.5.3 and Lemma 3.5.13 and each non-leaf node can be proved from its children using only those inference rules.

We are now ready to prove the completeness theorem of the sequents of the form $D, \Gamma \to \Delta$, where Γ and Δ are sets of PC formulas and D is a definition.

Theorem 3.5.15 (Completeness for one definition in the antecedent) *Let Γ and Δ be sets of PC formulas and D a definition. If $\models D, \Gamma \to \Delta$, then $D, \Gamma \to \Delta$ is provable in* **LPC(ID)**.

Proof. First, a reduction tree is constructed from the root $D, \Gamma \to \Delta$. Every leaf of the reduction tree must be an axiom or a sequent of the form $D, \Pi \to \Sigma$, where Π and Σ are (possibly empty) sets of propositional atoms satisfying that

1. Π and Σ have no atom in common, and
2. when Σ is not empty, Π or Σ contains at least one defined atom of D.

By (c) of Proposition 3.5.14, if $\models D, \Gamma \to \Delta$, then $\models D, \Pi \to \Sigma$. Hence, if Σ is empty, by Lemma 3.5.11, it is obtained that $D, \Pi \to \Sigma$ is provable in **LPC(ID)**. If Σ is not empty, by Lemma 3.5.12, the $\neg$ rules and the weakening rules, $D, \Pi \to \Sigma$ is provable in **LPC(ID)**. Extending for every leaf $D, \Pi \to \Sigma$ the branch that ends in that leaf with the proof tree for that leaf, yields an **LPC(ID)**-proof for $D, \Gamma \to \Delta$.

LPC(ID) remains complete for sequents of the form $D_1, \ldots, D_n, \Gamma \to \Delta$, where Γ and Δ are sets of PC formulas and multiple definitions are allowed in the antecedent.

Lemma 3.5.16 *Let $D_1, \ldots, D_n$ be definitions and Γ an arbitrary consistent set of literals. If $\models D_1, \ldots, D_n, \Gamma \rightarrow \bot$, then $D_1, \ldots, D_n, \Gamma \rightarrow \bot$ is provable in* **LPC(ID)**.

Proof. Let Γ' be an arbitrary extension of Γ such that for every $D_i \in \{D_1, \ldots, D_n\}$ and every open atom Q of D_i, either $Q \in \Gamma'$ or $\neg Q \in \Gamma'$. First, we want to show that $D_1, \ldots, D_n, \Gamma' \rightarrow \bot$ is provable in **LPC(ID)**. It holds that $\models D_1, \ldots, D_n, \Gamma' \rightarrow \bot$ because $\models D_1, \ldots, D_n, \Gamma \rightarrow \bot$. Consider the set Γ'' of all open literals of all definitions $D_1, \ldots, D_n$ in Γ'. We distinguish between the case where $\models D_1, \ldots, D_n, \Gamma'' \rightarrow \bot$ and the case where $\not\models D_1, \ldots, D_n, \Gamma'' \rightarrow \bot$.

- In the first case where $\models D_1, \ldots, D_n, \Gamma'' \rightarrow \bot$, we distinguish between the subcase where there exists at least one $D_i \in \{D_1, \ldots, D_n\}$ such that $\models D_i, \Gamma'' \rightarrow \bot$ and the subcase where for every $D_i \in \{D_1, \ldots, D_n\}$ it holds that $\not\models D_i, \Gamma'' \rightarrow \bot$.
  - In the first subcase, $\models D_i, \Gamma'' \rightarrow \bot$, hence by Lemma 3.5.11, $D_i, \Gamma'' \rightarrow \bot$ is provable in **LPC(ID)**. Then by using the left weakening rule, we conclude that $D_1, \ldots, D_n, \Gamma' \rightarrow \bot$ is provable in **LPC(ID)**.
  - In the other subcase, it holds that $\not\models D_i, \Gamma'' \rightarrow \bot$ for every $D_i \in \{D_1, \ldots, D_n\}$. Thus, for every $D_i \in \{D_1, \ldots, D_n\}$, there exists a unique two-valued well-founded model I_i of D_i such that $I_i \models D_i$ and $I_i \models \bigwedge \Gamma''$. Because $D_1 \wedge \ldots \wedge D_n \wedge \bigwedge \Gamma''$ is unsatisfiable, for some I_i and I_j such that $i \neq j$ and for some defined literal L, it can be implied that $I_i \models L$ and $I_j \models \neg L$. Thus, we have that $\models D_i, \Gamma'' \rightarrow L$ and $\models D_j, \Gamma'' \rightarrow \neg L$. Therefore, by Lemma 3.5.12, it is concluded that both $D_i, \Gamma'' \rightarrow L$ and $D_j, \Gamma'' \rightarrow \neg L$ are provable in **LPC(ID)**. Then we can use the left weakening rule, the left $\neg$ rule and the cut rule to show that $D_1, \ldots, D_n, \Gamma' \rightarrow \bot$ is provable in **LPC(ID)**.
- In the other case where $\not\models D_1, \ldots, D_n, \Gamma'' \rightarrow \bot$, hence there exists a unique two-valued interpretation I such that $I \models D_1 \wedge \ldots \wedge D_n \wedge \bigwedge \Gamma''$. Because $\not\models D_1, \ldots, D_n, \Gamma'' \rightarrow \bot$, for each $D_i \in \{D_1, \ldots, D_n\}$, it holds that $\not\models D_i, \Gamma'' \rightarrow \bot$ and hence, there exists a unique two-valued well-founded model I_i of D_i such that $I_i \models D_i$ and $I_i \models \bigwedge \Gamma''$. Therefore, for each D_i and each defined atom $P \in Def(D_i)$, $P^{I_i} = P^I$. Since $D_1 \wedge \ldots \wedge D_n \wedge \bigwedge \Gamma''$ is satisfiable but $D_1 \wedge \ldots \wedge D_n \wedge \bigwedge \Gamma'$ is unsatisfiable, it can be implied that for some defined literal L' in Γ', $\models D_1, \ldots, D_n, \Gamma'' \rightarrow \neg L'$. Assume that L' is a defined literal of D_i. Because $L'^{I_i} = L'^{I} = \mathbf{f}$, we have that $\models D_i, \Gamma'' \rightarrow \neg L'$. By Lemma 3.5.12 and the left weakening rule, $D_i, \Gamma' \rightarrow \neg L'$ is provable in **LPC(ID)**. It is obvious that $D_i, \Gamma' \rightarrow L'$ is an axiom because L' is a literal in Γ'. Then we can use the left weakening rule, the left $\neg$ rule and the cut rule to show that $D_1, \ldots, D_n, \Gamma' \rightarrow \bot$ is provable in **LPC(ID)**.

Given that the sequents $D_1, \ldots, D_n, \Gamma' \to \bot$ are provable in **LPC(ID)** for all extensions Γ' of Γ, by using the right $\neg$ rule and the cut rule on all $D_1, \ldots, D_n, \Gamma' \to \bot$, we can construct an **LPC(ID)**-proof for $D_1, \ldots, D_n, \Gamma \to \bot$.

Lemma 3.5.17 *Let $D_1, \ldots, D_n$ be definitions and let Γ and Δ be sets of atoms. If $\models D_1, \ldots, D_n, \Gamma \to \Delta$, then $D_1, \ldots, D_n, \Gamma \to \Delta$ is provable in* **LPC(ID)**.

Proof. The proof is trivial if $D_1, \ldots, D_n, \Gamma \to \Delta$ is an axiom, hence we assume that $D_1, \ldots, D_n, \Gamma \to \Delta$ is not an axiom, i.e., $\Gamma \cap \Delta = \emptyset$. Because $\Gamma, \neg\Delta$ is a consistent set of literals and $\models D_1, \ldots, D_n, \Gamma, \neg\Delta \to \bot$, by the previous lemma, we have that $D_1, \ldots, D_n, \Gamma, \neg\Delta \to \bot$ is provable in **LPC(ID)**. Then by the $\neg$ rules and the cut rule, we can conclude that $D_1, \ldots, D_n, \Gamma \to \Delta$ is provable in **LPC(ID)**.

The following completeness theorem of the sequents with multiple definitions in the antecedent is an immediate consequence of Lemma 3.5.17 and the reduction tree for sequents.

Theorem 3.5.18 (Completeness for multiple definitions in the antecedent) *Let Γ and Δ be sets of PC formulas and $D_1, \ldots, D_n$ definitions. If $\models D_1, \ldots, D_n, \Gamma \to \Delta$, then $D_1, \ldots, D_n, \Gamma \to \Delta$ is provable in* **LPC(ID)**.

Then we have the following main completeness theorem.

Theorem 3.5.19 (Completeness) *If $\models \Gamma \to \Delta$ and all definitions occurring either negatively in Γ or positively in Δ are total, then $\Gamma \to \Delta$ is provable in* **LPC(ID)**.

Proof. Let $\Gamma \to \Delta$ be a valid sequent such that any definition which occurs either negatively in Γ or positively in Δ is total and let T_S be a reduction tree with root $\Gamma \to \Delta$. Then by (c) of Proposition 3.5.14, all leaves of T_S are valid. Since all leaves are of the form $D_1, \ldots, D_n, \Pi \to \Sigma$ where Π and Σ are sets of atoms and $D_1, \ldots, D_n$ are definitions, it follows from Theorem 3.5.18 that they are provable in **LPC(ID)**. Hence, by (b) of Proposition 3.5.14, $\Gamma \to \Delta$ is provable in **LPC(ID)**.

3.6 Cut-elimination

An important property of the propositional sequent calculus and many other formal proof systems is the cut-elimination. The cut-elimination theorem states that any judgment that possesses a proof in the sequent calculus that makes use of the cut rule also possesses a cut-free proof, that is, a proof that does not make

use of the cut rule, i.e., removing the cut rule from the system does not lead to incompleteness. The cut-elimination theorem has many, rich consequences, e.g., once a proof system is shown to have a cut-elimination theorem, it is normally immediate that the system is consistent. Cut-elimination is also important in the development of goal-directed automated theorem proving systems. In a goal-directed automated proof, it is necessary to specify the "cut formula" that appears in the premises by introducing a new symbol, and it is unlikely that unification could determine such cut formulas. By cut-elimination, this rule is not needed to obtain a complete theorem prover.

For proof systems formulated in the sequent calculus, analytic proofs are those proofs that do not use cut. Typically such a proof will be longer. It was demonstrated in [22] that there was a derivation that could be completed in a page using cut, but whose analytic proof could not be completed in the lifespan of the universe.

The cut-elimination theorem does not hold for **LPC(ID)**. We give a counter-example to state that the cut-elimination is not applicable in **LPC(ID)** as follows.

Example 3.6.1 *Consider the definition*

$$D_1 = \left\{ \; P \leftarrow O_1 \; \right\}$$

and the definition

$$D_2 = \left\{ \; P \leftarrow O_2 \; \right\}.$$

Then $D_1, D_2, O_1, \neg O_2 \rightarrow \bot$ *can be proved is* **LPC(ID)** *as follows:*

$$\dfrac{\dfrac{\dfrac{O_1 \rightarrow O_1}{D_1, O_1 \rightarrow P}\ \textit{right definition rule}}{D_2, \neg O_2, D_1, O_1 \rightarrow P}\ \textit{left weakening} \qquad \dfrac{\dfrac{\dfrac{\neg P^{\triangleright}, \neg O_2 \rightarrow \neg O_2}{P, D_2, \neg O_2 \rightarrow \bot}\ \textit{left definition rule}}{D_1, O_1, P, D_2, \neg O_2 \rightarrow \bot}\ \textit{left weakening}}{D_1, D_2, O_1, \neg O_2 \rightarrow \bot}\ \textit{cut}$$

It can be proved that the use of the cut rule cannot be avoided for this example, which leads to the following property of **LPC(ID)**.

Proposition 3.6.2 *Cut is not eliminable in* **LPC(ID)**.

Proof. We prove that the cut rule is not eliminable in the derivation in Example 3.6.1. In a proof tree for $D_1, D_2, O_1, \neg O_2 \rightarrow \bot$, both D_1 and D_2 must be used in an application of the right definition or left definition rule. For if not, all occurrences of them could be removed from the proof tree, yielding a correct

proof tree with a non-valid root. Therefore, at some point in the proof tree, P must appear as an atomic formula in the antecedent or the succedent of a sequent. But in the root, P does only occur inside D_1 and D_2. So the other occurrences of P must have been removed by some rule. There are only six kinds of rules that remove formulas: the right definition rule, the left definition rule, the definition introduction rule, the non-total definition rule, the contraction rules and the cut rule. Both the left and right definition rules using D_1 or D_2 will introduce new occurrences of P. The definition introduction rule will introduce a definition in the succedent, which can only be removed by the cut rule. The non-total definition rule can only remove the occurrences of the renamed symbols rather than P. And the contraction rules can just remove the multiple occurrences of the same formulas in the antecedent or the succedent of a sequent. Hence, the cut rule is the only rule to remove the occurrences of P.

3.7 Complexity results

In this section, we provide some simple complexity results for PC(ID), which may give some helpful insight into the reasoning problems in PC(ID).

Proposition 3.7.1 *The satisfiability problem of PC(ID) is* NP*-complete.*

Proof. (Membership) Well-founded models can be computed in polynomial time, e.g., in quadratic time using an algorithm by Van Gelder [72]. It is easy to define an algorithm that uses such a well-founded semantics algorithm and finds models satisfying PC(ID) theories in polynomial time on a non-deterministic Turing machine.

(Hardness) Any satisfiability problem of propositional logic is trivially also a satisfiability problem of PC(ID), and the satisfiability problem of propositional logic itself is NP-complete.

Recall Definition 2.4.10 in Chapter 2 of totality of a definition D with respect to a theory T: for each $I \models \bigwedge T$, the well-founded model of D extending $I|_{Open(D)}$ must be two-valued. Deciding totality is an interesting problem, not least because we cannot even formulate an inference rule to prove totality of a propositional inductive definition in the context of a certain set of PC(ID) formulas.

Proposition 3.7.2 *Deciding whether a given propositional inductive definition is total with respect to a given propositional theory is a* co-NP*-complete problem.*

Proof. (Membership) Let D be a propositional inductive definition and T a propositional theory. Any interpretation I such that $I \models \bigwedge T$ and the well-founded model of D extending $I|_{Open(D)}$ is not two-valued, is a certificate for the non-totality of D with respect to T. Both checking whether $I \models \bigwedge T$ and whether the well-founded model of D extending $I|_{Open(D)}$ is two-valued can be done in polynomial time.

(Hardness) Consider the definition $D = \{ P \leftarrow \neg P \wedge T \}$. D is total with respect to the empty theory if and only if T is unsatisfiable. Thus we have found an instance of our decision problem that is equivalent to a co-NP-hard decision problem, namely the unsatisfiability problem for propositional logic.

3.8 Conclusions and related work

In this chapter, we have introduced a proof system **LPC(ID)** for PC(ID) based on the sequent calculus for propositional logic. We have established the soundness and completeness theorems for **LPC(ID)**. We have given a counter-example to cut-elimination for **LPC(ID)**. We also have provided some complexity results for PC(ID).

Related work is provided by Hagiya and Sakurai in [83]. They proposed to interpret a (stratified) logic program as iterated inductive definitions of Martin-Löf [119] and developed a proof theory which is sound with respect to the perfect model, and hence, the well-founded semantics of logic programming. A formal proof system based on tableau methods for analyzing computations in Answer Set Programming (ASP) was given as well by Gebser and Schaub [70]. As shown in [114], ASP is closely related to FO(ID). Their approach furnishes declarative and fine-grained instruments for characterizing operations as well as strategies of ASP-solvers and provides a uniform proof-theoretic framework for analyzing and comparing different algorithms, which is the first of its kind for ASP.

Chapter 4

LFO(ID,SLFP): a proof system for FO(ID) based on least fixpoint logic

4.1 Introduction

The initial motivation for the research of this chapter is to extend Gentzen's sequent calculus **LK** [74, 172] to obtain a proof system for FO(ID). We intend to build a sequent calculus for FO(ID) which is sound in general and complete for a useful subclass of FO(ID). The main challenge in building such a calculus is the representation and inference of unfounded sets [56]. In our approach to this problem, we represent unfounded sets by least fixpoint expressions borrowed from SLFP [40]. Compton described stratified least fixpoint logic (SLFP) in [40], which is a logic with a least fixpoint operator and characterizes the expressibility of stratified logic programs. He used Gentzen's sequent calculus to investigate deductive inference methods for SLFP and proved the soundness and completeness of such a proof system.

In our deductive system, we use Compton's inference system for SLFP. Our strategy consists of two steps. The first is to integrate least fixpoint expressions into FO(ID), leading to the logic FO(ID,SLFP). In the logic FO(ID,SLFP), we then build a proof calculus, which extends Compton's sequent calculus for SLFP with inference rules for the inductive definitions of FO(ID). By restricting input to FO(ID), this proof calculus yields a proof system for FO(ID) then. However, in proofs of FO(ID) formulas, some inference rules may introduce least fixpoint

expressions, and thus, proofs may contain expressions of FO(ID,SLFP).

We introduce the logic FO(ID,SLFP), which is an extension of FO with inductive definitions and stratified least fixpoint expressions in Section 4.2. We also introduce a normal form of inductive definitions in Section 4.2. In Section 4.3, we give the proof rules of **LFO(ID,SLFP)**, which are the rules of Gentzen's original sequent calculus for classical first-order logic, augmented with rules for stratified least fixpoint expressions, rules for introducing defined predicates on the left and right of sequents, and a rule for introducing definitions on the right of sequents. In Section 4.4, we show that our proof system is sound for a slightly restricted fragment of FO(ID,SLFP). In Section 4.5, we introduce a fragment of FO(ID,SLFP), namely $\mathcal{S}$FO(ID,SLFP), and show that our proof system is complete for this fragment. We finish with conclusions and related work in Section 4.6.

This work is a step forward in the development of domain independent deductive reasoning for FO(ID). This form of inference is similar as that in Description Logics or classical logic but differs from Answer Set Programming as the latter imposes Domain Closure. Our deductive calculus and the completeness result for a fragment of FO(ID), can be viewed as a step in developing tools similar to those in the field of description logics. In a similar aim, a decidable guarded fragment of FO(ID) was described recently in [177].

The first topic for future work is the development of tools to check the correctness of the outputs generated by FO(ID) model generators such as IDP [116, 115, 118]. Given an FO(ID) theory T and a finite domain D as input, a model generator outputs a model for T with domain D or concludes that T is unsatisfiable in D. In the former case, an independent *model checker* can be used to check whether the output is indeed a model of T. However, when the model generator concludes that T is unsatisfiable in D, it is less obvious how to check the correctness of this answer. A similar problem arises in SAT, where certain solvers can output a trace of a failed computation in the form of a resolution proof [184]. An independent *proof checker* can then be used to check this formal proof. For the future, we aim to study how to transform a trace of the computation of a model generator such as IDP into a proof of our proof system. Model and proof checkers can be a great help to detect bugs in model generators.

Also we intend to develop semi-automated and automated theorem proving systems for FO(ID) from the current deductive system for FO(ID) based on the least fixpoint logic, which may have more applications and implementations.

4.2 Preliminaries

First, we give an introduction to a normal form of definitions in Section 4.2.1. We define the logic FO(ID,SLFP), which is the extension of FO with both inductive definitions and stratified least fixpoint expressions in Section 4.2.2. And then we present the semantics of FO(ID,SLFP) in Section 4.2.3.

4.2.1 Normal form

A formula is in *negation normal form* (NNF) if negation occurs only immediately above elementary propositions, and $\{\neg, \vee, \wedge\}$ are the only allowed Boolean connectives. An arbitrary formula can be transformed into an equivalent formula in NNF using De Morgan's laws to push negation inside, and eliminating double negations. This process can be represented using the following *rewrite rules*:

- $\neg(\forall x\varphi) \rightarrow \exists x\neg\varphi$
- $\neg(\exists x\varphi) \rightarrow \forall x\neg\varphi$
- $\neg\neg\varphi \rightarrow \varphi$
- $\neg(\varphi_1 \wedge \varphi_2) \rightarrow (\neg\varphi_1) \vee (\neg\varphi_2)$
- $\neg(\varphi_1 \vee \varphi_2) \rightarrow (\neg\varphi_1) \wedge (\neg\varphi_2)$

A formula in negation normal form can be put into the stronger *conjunctive normal form* or *disjunctive normal form* by applying the *distributivity laws*.

Recall that a literal is an atom $P(\overline{t})$ or its negation $\neg P(\overline{t})$, called respectively a positive and a negative literal. For a literal l, we identify $\neg\neg l$ with l.A predicate symbol P has a negative (positive) occurrence in formula φ if P has an occurrence in the scope of an odd (even) number of occurrences of the negation symbol $\neg$ in φ.

An arbitrary definition D can be transformed into an equivalent definition such that each $P \in Def(D)$ is defined by exactly one rule, denoted by $\forall\overline{x}(P(\overline{x}) \leftarrow \varphi_P)$. We refer to this rule as *the rule defining* P. φ_P will be used to denote the body of the rule defining P. The transformation is obtained by merging all rules with the same head predicate into one rule as follows: $\{\forall\overline{x}(P(\overline{x}) \leftarrow \varphi_1), \ldots, \forall\overline{x}(P(\overline{x}) \leftarrow \varphi_n)\}$ becomes $\forall\overline{x}(P(\overline{x}) \leftarrow \varphi_1 \vee \ldots \vee \varphi_n)$, for all defined predicates P, and thus $\varphi_P = \varphi_1 \vee \ldots \vee \varphi_n$. Without loss of generality, in what follows we assume that there is only one rule $\forall\overline{x}(P(\overline{x}) \leftarrow \varphi_P)$ in a definition D for every $P \in Def(D)$.

4.2.2 Syntax of FO(ID,SLFP)

A vocabulary Σ consists of Σ^c and Σ^v where Σ^c consists of (countable) predicate constants and function symbols while Σ^v is a (countable) set of predicate variables. Object symbols and propositional symbols are 0-ary function symbols, respectively predicate symbols. Terms and FO formulas of Σ are defined as usual, and are built inductively from object symbols, function symbols, predicate symbols and logical connectives and quantifiers. A *literal* is an atom $P(t_1, \ldots, t_n)$ or its negation $\neg P(t_1, \ldots, t_n)$, called respectively a *positive* or a *negative* literal. For a literal l, we identify $\neg\neg l$ with l. A predicate symbol P has a negative (positive) occurrence in a formula φ if P has an occurrence in the scope of an odd (even) number of occurrences of the negation symbol $\neg$. A *definition* over Σ is a set of rules of the form $\forall \overline{x}(P(\overline{x}) \leftarrow \varphi)$, where $\overline{x}$ is a tuple of object variables over Σ^c, P is a predicate constant over Σ^c and φ is an FO formula over Σ^c such that all the free variables of φ occur in $\overline{x}$.

Now we are ready to define the formulas of FO(ID,SLFP).

Definition 4.2.1 *An* FO(ID,SLFP) formula *over* Σ *is defined by the following induction:*

- *If X is an n-ary predicate symbol (predicate constant or predicate variable) and $t_1, \ldots, t_n$ are terms then $X(t_1, \ldots, t_n)$ is a formula.*
- *If D is a definition then D is a formula.*
- *If ψ is a formula containing no free predicate variables, then $\neg\psi$ is a formula.*
- *If φ, ψ are formulas, then so are $\varphi \wedge \psi, \varphi \vee \psi, \varphi \supset \psi$ and $\varphi \equiv \psi$.*
- *If ψ is a formula and x is an object variable, then $\exists x\psi$ is a formula.*
- *If ψ is a formula containing no free predicate variables and x is an object variable, then $\forall x\psi$ is a formula.*
- *If $\psi, \theta_1, \ldots, \theta_n$ are formulas containing no definitions, each X_i is a k_i-ary predicate variable ($1 \leq i \leq n$) and each $\overline{x_i}$ is a tuple of k_i object variables ($1 \leq i \leq n$), then $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1, \ldots, \theta_n)]\psi$ is a formula and called a* stratified least fixpoint expression.

Example 4.2.2 *A Datalog query about membership of a pair (c, d) in the reflexive, transitive closure of a binary relation E is described by a stratified least fixpoint expression as:*

$$[LFP_{X(x,y)}(x = y \vee \exists z(X(x, z) \wedge E(z, y)))]X(c, d).$$

Note that the subformulas $\psi, \theta_1, \ldots, \theta_n$ of a stratified least fixpoint expression $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1, \ldots, \theta_n)]\psi$ may not contain definitions, but may contain stratified least fixpoint expressions. Indeed, nesting of stratified least fixpoint expressions is allowed in FO(ID,SLFP), but nesting of definitions is not. All subformulas $\psi, \theta_1, \ldots, \theta_n$ of an unnested stratified least fixpoint expression contain only positive occurrences of predicate variables. It is also worth mentioning that the unnested stratified least fixpoint expression $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1, \ldots, \theta_n)]\psi$, where $\theta_1, \ldots, \theta_n, \psi$ may not contain stratified least fixpoint expressions, corresponds exactly to the second-order ID-Logic formula $\exists X_1, \ldots, X_n(\left\{ \begin{array}{c} \forall \overline{x_1}(X_1(\overline{x_1}) \leftarrow \theta_1) \\ \vdots \\ \forall \overline{x_n}(X_n(\overline{x_n}) \leftarrow \theta_n) \end{array} \right\} \wedge \psi)$. However, such a correspondence does not hold for nested stratified least fixpoint expressions since only FO formulas are allowed as bodies of rules in definitions.

Definition 4.2.3 *Each occurrence of an object variable x in formulas $\exists x\varphi$ and $\forall x\varphi$ is* bound. *All occurrences of variables $\overline{x_1}, \ldots, \overline{x_n}$ in the subformulas $\theta_1, \ldots, \theta_n$ of a stratified least fixpoint expression $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1, \ldots, \theta_n)]\psi$ are bound. Each occurrence of a predicate variable X_i inside subformulas $\psi, \theta_1, \ldots, \theta_n$ of a stratified least fixpoint expression $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1, \ldots, \theta_n)]\psi$ is bound. All other occurrences of an object or predicate variable is* free. *Let us denote $free(\varphi)$ the set of object and predicate variables with a free occurrence in φ.*

An FO formula is an FO(ID,SLFP) formula without definitions and stratified least fixpoint expressions. An FO(ID) formula is an FO(ID,SLFP) formula without stratified least fixpoint expressions, and an SLFP formula is one without definitions. An FO(ID,SLFP) *sentence* is a formula without free variables. An FO(ID,SLFP) *theory* is a set of FO(ID,SLFP) sentences.

4.2.3 Semantics of FO(ID,SLFP)

The semantics of FO(ID,SLFP) is an integration of standard FO semantics, the well-founded semantics of definitions and the semantics of SLFP. Since both the standard FO semantics and the well-founded semantics of definitions were presented specifically in Chapter 2, here we only focus on the semantics of SLFP (see [40]).

Let φ be an SLFP formula. The notation $\varphi(P/\rho)$ means that all free occurrences of the predicate variable P in φ are replaced by the formula ρ. (To be precise, we should specify a sequence of k distinct object variables in ρ, where k is the arity of P; the correspondence between object variables of P and object variables of ρ will

always be clear from context.) All uses of this notation are subject to the proviso that free occurrences of variables in P remain free wherever ρ is substituted.

We now give the semantics of SLFP formulas. As usual, we define by induction on φ the relation $I \models \varphi[\alpha]$ (a structure I satisfies φ at α), where α is a variable assignment in I that interprets each free variable in φ. More precisely, suppose φ has free predicate variables $P_1, \ldots, P_k$, with respective arities $j_1, \ldots, j_k$, and free object variables $x_1, \ldots, x_l$. Fix a structure I. An assignment α for φ can be represented as a sequence $(R_1, \ldots, R_k, a_1, \ldots, a_l)$, where each R_i is a j_i-ary relation on $dom(I)$ and each a_i is an element of $dom(I)$. With φ we will associate a function F_φ mapping sequences $(R_1, \ldots, R_k)$ to l-ary relations on $dom(I)$:

$$F_\varphi(R_1, \ldots, R_k) = \{(a_1, \ldots, a_l) \mid I \models \varphi[R_1, \ldots, R_k, a_1, \ldots, a_l]\}.$$

Simultaneously with our definition of satisfaction, it can be shown that F_φ is *continuous*; i.e., that

$$\bigcup_{\alpha<\lambda} F_\varphi(R_{1\alpha}, \ldots, R_{k\alpha}) = F_\varphi(\bigcup_{\alpha<\lambda} R_{1\alpha}, \ldots, \bigcup_{\alpha<\lambda} R_{k\alpha})$$

for all chains $(R_{i\alpha} \mid \alpha < \lambda)$ of j_i-ary relations. Notice that if F_φ is continuous, it is *monotone* as well. By a continuous (or monotone) formula, we mean a formula φ such that F_φ is continuous (or monotone).

$I \models \varphi[\alpha]$ is defined in the usual way when φ is an atom, a disjunction, conjunction, negation or quantified formula and it is not difficult to see that φ is continuous. (Notice, however, that it is crucial that negations are not applied to formulas with free predicate variables.)

For convenience, we define $I \models \varphi[\alpha]$ where φ is of the form $[LFP_{X(\overline{x})}(\theta)]\psi$ with single predicate variable X and formula θ in the least fixpoint expression. (Actually, we just need to make obvious modifications to define $I \models \varphi[\alpha]$ where φ is of the form $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1, \ldots, \theta_n)]\psi$ (see [40])). Assume that θ and ψ are continuous and their truth values have been defined for the assignment α. Let the assignments to variables other than X and $\overline{x} = (x_1, \ldots, x_k)$ be given by the assignment α. We thereby obtain from F_θ a continuous mapping G from k-ary relations to k-ary relations. G is monotone and hence has a least fixpoint by the Least Fixpoint Theorem [174].

The well-known construction of the least fixpoint of a monotone function is as follows. Let $G^0(R) = R, G^{\beta+1}(R) = G(G^\beta(R))$ and if β is a limit ordinal, $G^\beta(R) = \bigcup_{\gamma<\beta} G^\gamma(R)$. By induction $G^\beta(\emptyset) \subseteq G^\gamma(\emptyset)$ whenever $\beta < \gamma$. On each structure there is a smallest ordinal κ, called the *closure ordinal* of the inductive expression $LFP_{X(\overline{x})}(\theta)$, such that $G^\beta(\emptyset) = G^\kappa(\emptyset)$ whenever $\beta \geq \kappa$. $G^\kappa(\emptyset)$ is the least fixpoint of G. Since G is continuous, it follows that $\kappa \leq \omega$ (see [40]). Thus, $I \models \varphi[\alpha]$ holds in case $I \models \psi[\alpha']$, where α' is identical to α except that it assigns $G^\omega(\emptyset)$ to X.

It will be useful to define, for each nonnegative integer m, the formula

$$[LFP_{X(\overline{x})}(\theta)]_m\psi.$$

$I \models [LFP_{X(\overline{x})}(\theta)]_m\psi[\alpha]$ holds just in case $I \models \psi[\alpha'']$, where α'' is identical to α except that it assigns $G^m(\emptyset)$ to X. We regard this formula as an abbreviation. Construct a sequence of formulas $\rho_0, \rho_1, \rho_2, \ldots$, where ρ_0 is $\bot$ and ρ_{m+1} is the formula $\theta(X/\rho_m)$. Then $[LFP_{X(\overline{x})}(\theta)]_m\psi$ is an abbreviation for $\psi(X/\rho_m)$. Call this formula φ_m.

Since continuity is preserved by composition, each of the functions F_{φ_m} is continuous. Moreover, the sequence $F_{\varphi_0}, F_{\varphi_1}, F_{\varphi_2}, \ldots$ is a chain (in the partial order of function dominance) with supremum F_φ. Since the supremum of a chain of continuous functions is continuous, F_φ is continuous (see Theorem 4.18 of [110]). It follows that $[LFP_{X(\overline{x})}(\theta)]\psi$ is equivalent to the infinite disjunction

$$\bigvee_{m\in\omega}[LFP_{X(\overline{x})}(\theta)]_m\psi.$$

We summarize our observations in the following theorem.

Theorem 4.2.4 *The following hold for SLFP.*

1. *All formulas are continuous (and hence monotone).*
2. *The closure ordinal of any inductive expression is at most ω.*
3. *$[LFP_{X(\overline{x})}(\theta)]\psi$ is equivalent to $\bigvee_{m\in\omega}[LFP_{X(\overline{x})}(\theta)]_m\psi$.*

This theorem is due to [139]. Part 2 of this theorem was observed by Aczel [2] for systems of existential inductive definitions. Blass and Gurevich observed that part 2 of this theorem is true for existential least fixpoint formulas in [20]. The formula $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$ is defined analogously to $[LFP_{X(\overline{x})}(\theta)]_m\psi$. Moreover, it is shown in [40] that $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$ is equivalent to the infinite disjunction $\bigvee_{m\in\omega}[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$.

The following notations will be used frequently in the remainder of this chapter. Given two disjunct first-order vocabularies Σ and Σ', a Σ-interpretation I and a Σ'-interpretation I', the $\Sigma\cup\Sigma'$-interpretation mapping each element e of Σ to e^I and each $e \in \Sigma'$ to $e^{I'}$ is denoted by $I + I'$. When $\Sigma' \subseteq \Sigma$, we denote the restriction of a Σ-interpretation I to the symbols of Σ' by $I|_{\Sigma'}$. For a Σ-interpretation I, a truth value v and a domain atom $P(\overline{a})$, we denote by $I[P(\overline{a})/v]$ the Σ-interpretation that assigns v to $P(\overline{a})$ and corresponds to I for all other domian atoms. We extend this notation to sets of domain atoms, i.e., for a Σ-interpretation I, a truth value v and a set of domain atoms $\mathcal{D}$, we denote by $I[\mathcal{D}/v]$ the Σ-interpretation that assigns v to every $P(\overline{a}) \in \mathcal{D}$ and corresponds to I for all other domain atoms.

4.3 The deductive system for FO(ID,SLFP)

In this section, we shall give the inference rules of the deductive system for FO(ID,SLFP), **LFO(ID,SLFP)**, presented in the sequent calculus style.

The inference rules for **LFO(ID,SLFP)** consist of all the inference rules of Gentzen's original sequent calculus **LK** for classical first-order logic, *SLFP* rules [40] and *definition* rules. The rules for classical first-order logic deal with the first-order part of FO(ID,SLFP) and are given as in Chapter 2.

Our deductive system **LFO(ID,SLFP)** is then obtained from **LK** by adding rules for introducing stratified least fixpoint expressions on the right and left of sequents, rules for introducing defined atoms on the right and left of sequents and a rule for introducing definitions on the right of sequents. The SLFP rules of **LFO(ID,SLFP)** consist of the *left SLFP* rule and the *right SLFP* rule. The definition rules of **LFO(ID,SLFP)** consist of the *right definition* rule, the *left definition* rule and the *definition introduction* rule.

SLFP rules

The inference rules for stratified least fixpoint expressions, which were given in [40], consist of the right SLFP rule and the left SLFP rule. Actually, the left SLFP rule is an infinitary inference rule with countably infinite number of premises. In [40], it is shown that SLFP is not compact. It follows that we must have some sort of infinitary rules in any complete sequent calculus for SLFP. Such infinitary rules are expressed in the form of

$$\frac{\Gamma_m \to \Delta_m \quad (m \in \omega)}{\Gamma \to \Delta}.$$

Now we are ready to give the SLFP rules as follows.

- left: $\dfrac{\Gamma, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \to \Delta \quad (m \in \omega)}{\Gamma, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \to \Delta}$;
- right: $\dfrac{\Gamma \to \Delta, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi}{\Gamma \to \Delta, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi}$

Notice that, as mentioned previously, the left stratified least fixpoint rule is infinitary: it has countably many premises while in the right stratified least fixpoint rule there is just one premise: m is a fixed nonnegative integer.

Right definition rule for P.

The *right definition rule* introduces defined atomic formulas in the succedents of sequents. It allows inferring the truth of a defined atom from a definition D and is therefore closely related to the first step for extending a well-founded induction. Let D be a definition and P a defined predicate of D. The right definition rule for P is given as follows.

$$\frac{\Gamma \to \Delta, \varphi_P(\overline{t})}{D, \Gamma \to \Delta, P(\overline{t})}$$

where Γ and Δ are arbitrary sets of FO(ID,SLFP) formulas.

We illustrate this inference rule with an example.

Example 4.3.1 *Consider the definition D of the transitive closure of a directed graph G as*

$$\left\{ \ \forall x, y(T(x,y) \leftarrow G(x,y) \vee \exists z(T(x,z) \wedge T(z,y))) \ \right\},$$

$\Gamma = \{G(a,b), G(b,c)\}$ *and* Δ *an empty set. The instance of the right definition rule for $T(a,c)$ is*

$$\frac{G(a,b), G(b,c) \to G(a,c) \vee \exists z(T(a,z) \wedge T(z,c))}{D, G(a,b), G(b,c) \to T(a,c)}.$$

Left definition rule.

The *left definition rule* introduces defined atoms in the antecedents of sequents. It allows inferring the falsity of defined atoms from a definition D and is therefore closely related to the second step for extending a well-founded induction.

We first introduce some notations. Given a set U of predicate symbols, let U' be a set consisting of new predicate symbols P' for each $P \in U$. The vocabulary Σ augmented with these new symbols is denoted by Σ'. Given an FO formula φ, φ' denotes the formula obtained by replacing all positive occurrences of every $P \in U$ in φ by P'. We call φ' the *renaming* of φ with respect to U. By $\neg\varphi'$, we mean $\neg(\varphi')$.

Let D be a definition and $U = \{P_1, \ldots, P_n\}$ a non-empty set of defined predicates of D. Let $\psi_1(\overline{x^1}), \ldots, \psi_n(\overline{x^n})$ be n arbitrary FO formulas over Σ or stratified least fixpoint expressions over Σ with $\overline{x^i}$ as many fresh variables as the arity of $P_i(\overline{x^i})$. Then the left definition rule for $(\{P_1(\overline{x^1}) \mid \psi_1(\overline{x^1})\}, \ldots, \{P_n(\overline{x^n}) \mid \psi_n(\overline{x^n})\})$ is given as follows.

Left definition rule for $\{P_i(\overline{x^i}) \mid \psi_i(\overline{x^i})\}_{i\in\{1,\ldots,n\}}$

$$\frac{\begin{array}{c}\Gamma, \forall\overline{x^1}(\psi_1(\overline{x^1}) \supset \neg P'_1(\overline{x^1})), \ldots, \forall\overline{x^n}(\psi_n(\overline{x^n}) \supset \neg P'_n(\overline{x^n})) \\ \rightarrow \Delta, \forall\overline{x^1}(\psi_1(\overline{x^1}) \supset \neg\varphi'_{P_1}(\overline{x^1})) \\ \vdots \\ \Gamma, \forall\overline{x^1}(\psi_1(\overline{x^1}) \supset \neg P'_1(\overline{x^1})), \ldots, \forall\overline{x^n}(\psi_n(\overline{x^n}) \supset \neg P'_n(\overline{x^n})) \\ \rightarrow \Delta, \forall\overline{x^n}(\psi_n(\overline{x^n}) \supset \neg\varphi'_{P_n}(\overline{x^n}))\end{array}}{D, \Gamma, \exists\overline{x^i}(\psi_i(\overline{x^i}) \wedge P_i(\overline{x^i})) \rightarrow \Delta}$$

where Γ and Δ are sets of FO(ID,SLFP) formulas over Σ and $\varphi'_{P_i}(\overline{x^i})$ is the renaming of $\varphi_{P_i}(\overline{x^i})$ with respect to U.

Actually, in the left definition rule, the set of defined atoms $\{P_1(\overline{x^1}) \mid \psi_1(\overline{x^1})\} \cup \ldots \cup \{P_n(\overline{x^n}) \mid \psi_n(\overline{x^n})\}$ is a symbolic representation of a candidate unfounded set of D.

We illustrate this inference rule with the following example.

Example 4.3.2 *Given the definition D of the transitive closure of a directed graph G as*

$$\left\{\ \forall x, y(T(x,y) \leftarrow G(x,y) \vee \exists z(T(x,z) \wedge T(z,y)))\ \right\},$$

$\Gamma = \{\neg\exists x G(x,a)\}$ and Δ an empty set. The instance of the left definition rule for $(\{T(x,y) \mid y = a\})$ is as follows:

$$\frac{\begin{array}{c}\neg\exists x G(x,a), \forall x, y(y = a \supset \neg T'(x,y)) \\ \rightarrow \forall x, y(y = a \supset \neg G(x,y) \wedge \neg\exists z(T'(x,z) \wedge T'(z,y)))\end{array}}{D, \neg\exists x G(x,a), \exists x, y(y = a \wedge T(x,y)) \rightarrow}.$$

Notice that the set $\{T(x,y) \mid y = a\}$ is the symbolic representation of an unfounded set of D.

Definition introduction rule for D.

The two definitional inference rules introduced so far, introduce a definition in the antecedent of the consequence. Hence, none of these rules can be used to infer that under certain conditions a definition holds. The *definition introduction rule* allows inferring the truth of a total definition from FO(ID,SLFP) formulas.

We first introduce some notations. Let D be a total definition. Denote by P^R a new defined predicate for each $P \in Def(D)$. Denote by Σ^R the vocabulary $\Sigma \cup \{P^R \mid P \in Def(D)\}$. Denote by D^R the definition over the new vocabulary Σ^R obtained by replacing each occurrence of each defined predicate P in D by P^R. Let Γ and Δ be sets of FO(ID,SLFP) formulas over the old vocabulary Σ.

The definition introduction rule for D is given as follows, where $P_1, \ldots, P_n$ are all defined predicates of D.

$$\frac{\Gamma, D^R \to \forall \overline{x^1}(P_1^R(\overline{x^1}) \equiv P_1(\overline{x^1})), \Delta; \ldots; \Gamma, D^R \to \forall \overline{x^n}(P_n^R(\overline{x^n}) \equiv P_n(\overline{x^n})), \Delta}{\Gamma \to \Delta, D}.$$

We illustrate this inference rule with an example.

Example 4.3.3 *Given a definition*

$$D = \left\{ \ \forall x(P(x) \leftarrow P(x) \wedge O(x)) \ \right\},$$

$\Gamma = \{\forall x O(x), \forall x \neg P(x)\}$ *and* Δ *an empty set. The instance of the definition introduction rule for* D *is as follows:*

$$\frac{D^R, \forall x O(x), \forall \neg P(x) \to \forall x(P^R(x) \equiv P(x))}{\forall x O(x), \forall x \neg P(x) \to D},$$

where $D^R = \left\{ \ \forall x(P^R(x) \leftarrow P^R(x) \wedge O(x)) \ \right\}$.

The inference rule proposed here has a definition in the succedent of its premise and hence, allows to infer the truth of a definition. Unfortunately, this rule is only sound given that the inferred definition is total. We will give an example to show that the definition introduction rule is not sound given that the inferred definition is non-total right after proving the soundness of this inference rule.

Proofs of LFO(ID,SLFP).

We now come to the notion of an **LFO(ID,SLFP)**-*proof* for a sequent.

Definition 4.3.4 *A* proof *in* **LFO(ID,SLFP)** *or an* **LFO(ID,SLFP)**-proof *for a sequent* S*, is a tree* T *of sequents with root* S*. Moreover, each leaf of* T *must be an axiom and for each interior node* S' *there exists an instance of an inference rule in* **LFO(ID,SLFP)** *such that* S' *is the consequence of that instance while the children of* S' *are precisely the premises of that instance.* T *is often called a* proof tree *for* S*. A sequent* S *is called* provable *in* **LFO(ID,SLFP)***, or* **LFO(ID,SLFP)**-provable*, if there is an* **LFO(ID,SLFP)**-*proof for it.*

Example 4.3.5 *Given the definition of the transitive closure of a directed graph* G*:*

$$D = \left\{ \ \forall x, y(T(x,y) \leftarrow G(x,y) \vee \exists z(T(x,z) \wedge T(z,y))) \ \right\},$$

an **LFO(ID,SLFP)**-*proof for* $D, G(a,b), G(b,c) \to T(a,c)$ *is as follows:*

$$\dfrac{\dfrac{\dfrac{\dfrac{G(a,b)\to G(a,b)}{G(a,b)\to G(a,b),\exists z(T(a,z)\wedge T(z,b))}}{G(a,b)\to G(a,b)\vee\exists z(T(a,z)\wedge T(z,b))}}{D,G(a,b)\to T(a,b)}\qquad \dfrac{\dfrac{\dfrac{\dfrac{G(b,c)\to G(b,c)}{G(b,c)\to G(b,c),\exists z(T(b,z)\wedge T(z,c))}}{G(b,c)\to G(b,c)\vee\exists z(T(b,z)\wedge T(z,c))}}{D,G(b,c)\to T(b,c)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{D,G(a,b),G(b,c)\to T(a,b)\wedge T(b,c)}{D,G(a,b),G(b,c)\to\exists z(T(a,z)\wedge T(z,c))}}{D,G(a,b),G(b,c)\to G(a,c),\exists z(T(a,z)\wedge T(z,c))}}{D,G(a,b),G(b,c)\to G(a,c)\vee\exists z(T(a,z)\wedge T(z,c))}}{D,D,G(a,b),G(b,c)\to T(a,c)}}{D,G(a,b),G(b,c)\to T(a,c)}$$

Example 4.3.6 *Given a definition*

$$D=\left\{\begin{array}{c}\forall x(P(x)\leftarrow P(x)\wedge O(x))\\ \forall x(Q(x)\leftarrow\neg P(x))\end{array}\right\},$$

an **LFO(ID,SLFP)***-proof for* $D,\forall xO(x)\to\forall xQ(x)$ *is as follows:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg P'(a)\to\neg P'(a)}{O(a),\neg P'(a)\to\neg P'(a)}}{O(a),\neg P'(a)\to\neg P'(a),\neg O(a)}}{O(a),\neg P'(a)\to\neg P'(a)\vee\neg O(a)}}{\forall xO(x),\forall x\neg P'(x)\to\neg P'(a)\vee\neg O(a)}}{\forall xO(x),\forall x\neg P'(x)\to\forall x(\neg P'(x)\vee\neg O(x))}}{D,\forall xO(x),\exists xP(x)\to}}{D,\forall xO(x)\to\forall x\neg P(x)}\qquad \dfrac{\dfrac{\dfrac{\dfrac{\neg P(b)\to\neg P(b)}{D,\neg P(b)\to Q(b)}}{D,\forall x\neg P(x)\to Q(b)}}{D,\forall\neg P(x)\to\forall xQ(x)}}{D,\forall xO(x),\forall\neg P(x)\to\forall xQ(x)}$$

$$\dfrac{D,\forall xO(x)\to\forall x\neg P(x)\qquad D,\forall xO(x),\forall\neg P(x)\to\forall xQ(x)}{D,\forall xO(x)\to\forall xQ(x)}$$

4.4 Soundness of LFO(ID,SLFP)

We now prove that inference rules of **LFO(ID,SLFP)** are sound, i.e., if all premises of an inference rule are valid then the consequence of that rule is valid. The main interesting cases are the SLFP rules and the definition rules. Although the proof is fairly standard in the remaining cases, we nevertheless treat them comprehensively here for the sake of completeness. We firstly give the soundness proof of every inference rule in the classical sequent calculus **LK** for first-order logic as follows.

Lemma 4.4.1 *Let $\mathcal{R}$ be an arbitrary inference rule in* **LK** *and I an interpretation. If all premises of $\mathcal{R}$ are true in I, then the conclusion of $\mathcal{R}$ is also true in I.*

Proof. We just need to check that each rule in **LK** has the desired property. We omit the cases of propositional rules here since the results of the soundness of them have been presented in the previous chapter.

- **Case (Left $\forall$):**
$$\frac{\varphi(t), \Gamma \to \Delta}{\forall x \varphi(x), \Gamma \to \Delta}.$$
Suppose that $I \models \forall x \varphi(x)$ and $I \models \bigwedge \Gamma$. It follows immediately that $I \models \varphi(t)$. Since $\varphi(t), \Gamma \to \Delta$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Right $\forall$):**
$$\frac{\Gamma \to \Delta, \varphi(x)}{\Gamma \to \Delta, \forall y \varphi(y)} \; x \notin free(\Gamma \cup \Delta).$$
Suppose that $I \models \bigwedge \Gamma$. Since $\Gamma \to \Delta, \varphi(x)$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, it holds that either $I \models \bigvee \Delta$ or $I \models \varphi(x)$. If $I \models \bigvee \Delta$, then we are done; otherwise $I \models \varphi(x)$ and since $x \notin free(\Gamma \cup \Delta)$, it holds that $I \models \forall y \varphi(y)$, as desired.

- **Case (Left $\exists$), (Right $\exists$):** Similar to the cases (Right $\forall$) and (Left $\forall$) above.

- **Case (Left $=$):**
$$\frac{\varphi(x/t), \Gamma \to \Delta}{t = u, \varphi[x/u], \Gamma \to \Delta}.$$
Suppose that $I \models t = u$, $I \models \varphi[x/u]$ and $I \models \bigwedge \Gamma$. Since $t^I = u^I$ and $I \models \varphi[x/u]$, it follows directly that $I \models \varphi[x/t]$. By the assumption that $\varphi[x/t], \Gamma \to \Delta$ is true in I and the facts that $I \models \varphi[x/t]$ and $I \models \bigwedge \Gamma$, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Right $=$):**
$$\frac{\Gamma \to \Delta, \varphi[x/t]}{t = u, \Gamma \to \Delta, \varphi[x/u]}.$$
Suppose that $I \models t = u$ and $I \models \bigwedge \Gamma$. Since $\Gamma \to \Delta, \varphi[x/t]$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, it holds that either $I \models \bigvee \Delta$ or $I \models \varphi[x/t]$. If $I \models \bigvee \Delta$, then we are done; otherwise $I \models \varphi[x/t]$ and since $t^I = u^I$, it holds that $I \models \varphi[x/u]$, as desired.

We now give the soundness proof of every SLFP rule.

Lemma 4.4.2 *Let $\mathcal{R}$ be an arbitrary SLFP rule in* **LFO(ID,SLFP)** *and I an interpretation. If all premises of $\mathcal{R}$ are true in I, then the conclusion of $\mathcal{R}$ is also true in I.*

Proof. We have to present that both left and right SLFP rules are sound.

- **Case (Left SLFP):**

$$\frac{\Gamma, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \rightarrow \Delta \quad (m \in \omega)}{\Gamma, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \rightarrow \Delta}.$$

  Suppose that $I \models \bigwedge \Gamma$ and $I \models [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$. By Theorem 4.2.4, we know that $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$ is equivalent to the infinite disjunction $\bigvee_{m\in\omega}[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$. It follows directly that $I \models \bigvee_{m\in\omega}[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$. Thus, by the assumption that $\Gamma, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \rightarrow \Delta \ (m \in \omega)$ are true in I and the fact that $I \models [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$ for some $m \in \omega$, it holds that $I \models \bigvee \Delta$, as desired.

- **Case (Right SLFP):**

$$\frac{\Gamma \rightarrow \Delta, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi}{\Gamma \rightarrow \Delta, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi}.$$

  Suppose that $I \models \bigwedge \Gamma$. Since $\Gamma \rightarrow \Delta, [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$ is true in I by the assumption and the fact that $I \models \bigwedge \Gamma$, it holds that either $I \models \bigvee \Delta$ or $I \models [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$ for some $m \in \omega$. If $I \models \bigvee \Delta$, then we are done; otherwise $I \models [LFP_{P_1(\overline{x_1}),\ldots,P_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$ for some $m \in \omega$ and thus, $I \models \bigvee_{m\in\omega}[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$. Since $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$ is equivalent to $\bigvee_{m\in\omega}[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi$, it holds that $I \models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$, as desired.

We now proceed to give the soundness proof of each definition rule.

Lemma 4.4.3 *Let I be a model of a definition D and P a defined predicate of D. $I \models P(\overline{t})$ iff $I \models \varphi_P(\overline{t})$.*

Proof. Because I is a model of D, there exists a terminal well-founded induction $(I^\xi)_{\xi\leq\alpha}$ for D with the limit I. Assume that $I \models \varphi(\overline{t})$. The sequence $(I^\xi)_{\xi\leq\alpha}$ is strictly increasing in precision, hence there is no $\xi \leq \alpha$ such that $\varphi(\overline{t})^{I^\xi} = \mathbf{f}$. As such, for every $\xi \leq \alpha$, $P(\overline{t})^{I^\xi} \neq \mathbf{f}$. Therefore, $P(\overline{t})^{I} \neq \mathbf{f}$ and because I is

two-valued, we have $P(\overline{t})^I = \mathbf{t}$. Assume that $I \models P(\overline{t})$. Thus, for some $\xi < \alpha$, $P(\overline{t})^{I^\xi} = \mathbf{u}$ and $P(\overline{t})^{I^{\xi+1}} = \mathbf{t}$. Hence $\varphi(\overline{t})^{I^\xi} = \mathbf{t}$. Because the sequence $(I^\xi)_{\xi \leq \alpha}$ is strictly increasing in precision, we have $\varphi(\overline{t})^I = \mathbf{t}$.

Lemma 4.4.4 (Soundness of the right definition rule) *If* $\models \Gamma \to \Delta, \varphi_P(\overline{t})$ *then* $\models D, \Gamma \to \Delta, P(\overline{t})$.

Proof. Assume that $\models \Gamma \to \Delta, \varphi_P(\overline{t})$ but $\not\models D, \Gamma \to \Delta, P(\overline{t})$. Then there exists a counter-interpretation I for $D, \Gamma \to \Delta, P(\overline{t})$ which satisfies $D, \bigwedge \Gamma, \neg \bigvee \Delta$ and $\neg P(\overline{t})$. It follows from the first assumption that $I \models \varphi_P(\overline{t})$, and hence, by Lemma 4.4.3, $I \models P(\overline{t})$, a contradiction.

Lemma 4.4.5 (Soundness of the left definition rule) *If* $\models \Gamma, \forall \overline{x^1}(\psi(\overline{x^1}) \supset \neg P'_1(\overline{x^1})), \ldots, \forall \overline{x^n}(\psi_n(\overline{x^n}) \supset \neg P'_n(\overline{x^n})) \to \Delta, \forall \overline{x^i}(\psi_i(\overline{x^i}) \supset \neg \varphi'_{P_i}(\overline{x^i}))$ *for every* $i \in \{1, \ldots, n\}$, *then* $\models D, \Gamma, \exists \overline{x^i}(\psi_i(\overline{x^i}) \wedge P_i(\overline{x^i})) \to \Delta$ *for each* $i \in \{1, \ldots, n\}$.

Proof. Assume that $\models \Gamma, \forall \overline{x^1}(\psi_1(\overline{x^1}) \supset \neg P'_1(\overline{x^1})), \ldots, \forall \overline{x^n}(\psi_n(\overline{x^n}) \supset \neg P'_n(\overline{x^n})) \to \Delta, \forall \overline{x^i}(\psi_i(\overline{x^i}) \supset \neg \varphi'_{P_i}(\overline{x^i}))$ for each $i \in \{1, \ldots, n\}$ but there exists at least one $P_i \in \{P_1, \ldots, P_n\}$ such that $\not\models D, \Gamma, \exists \overline{x^i}(\psi_i(\overline{x^i}) \wedge P_i(\overline{x^i})) \to \Delta$. Then there exists a model I of $D, \bigwedge \Gamma$ and $\neg \bigvee \Delta$ such that $I \models \exists \overline{x^i}(\psi_i(\overline{x^i}) \wedge P_i(\overline{x^i}))$. Thus, $I \models \psi_i(\overline{a}) \wedge P_i(\overline{a})$ for some tuple of domain elements $\overline{a}$. We select this $P_i(\overline{a})$ in the following way. Let $(I^\xi)_{\xi \leq \alpha}$ be a terminal well-founded induction for D with the limit $I^\alpha = I$. Let ξ be the smallest $\xi \leq \alpha$ such that for some $P_i \in \{P_1, \ldots, P_n\}$ and some tuple of domain elements $\overline{a}$, $\psi_i(\overline{a})^I = \mathbf{t}, P_i(\overline{a})^{I^\xi} = \mathbf{u}$ and $P_i(\overline{a})^{I^{\xi+1}} = \mathbf{t}$. By selection of ξ, there is a unique $P_j \in \{P_1, \ldots, P_n\}$ with its rule $\forall \overline{x}(P_j(\overline{x}) \leftarrow \varphi_{P_j}(\overline{x})) \in D$ and a unique tuple of domain elements $\overline{b}$ such that $P_j(\overline{b})^{I^\xi} = \mathbf{u}, \varphi_{P_j}(\overline{b})^{I^\xi} = \mathbf{t}, P_j(\overline{b})^{I^{\xi+1}} = \mathbf{t}$ and $\psi_j(\overline{b})^I = \mathbf{t}$. Consider this $P_j(\overline{b})$ and $\varphi_{P_j}(\overline{b})$.

On the one hand, it holds that $I^\xi \models \varphi_{P_j}(\overline{b})$. On the other hand, consider the interpretation $I' = I[\begin{array}{l} P'_i(\overline{a})/\mathbf{f} \text{ if } \psi_i(\overline{a})^I = \mathbf{t} \\ P'_i(\overline{a})/P_i(\overline{a})^I \text{ if } \psi_i(\overline{a})^I = \mathbf{f} \end{array}]$. Because Γ and Δ do not contain any new symbols, it is clear that I' satisfies $\bigwedge \Gamma$ and $\neg \bigvee \Delta$. Because of the construction of I' and the fact that $\psi_i(\overline{x^i})$ does not contain any new symbols, we have that $I' \models \forall \overline{x^i}(\psi_i(\overline{x^i}) \supset \neg P'_i(\overline{x^i}))$ for each $i \in \{1, \ldots, n\}$. Hence, by the first assumption, it holds that $I' \models \forall \overline{x^j}(\psi_j(\overline{x^j}) \supset \neg \varphi'_{P_j}(\overline{x^j}))$. Because $\psi_j(\overline{b})^I = \mathbf{t}$ and $\psi_j(\overline{b})$ does not contain any new symbols, it holds that $\psi_j(\overline{b})^{I'} = \mathbf{t}$. Thus, it is obtained that $I' \models \neg \varphi'_{P_j}(\overline{b})$. We want to derive a contradiction from $I' \models \neg \varphi'_{P_j}(\overline{b})$ and $I^\xi \models \varphi_{P_j}(\overline{b})$.

Observe that by our choice of ξ, for each $P_i \in \{P_1, \ldots, P_n\}$ and arbitrary tuple of domain elements $\overline{a}$, $P_i(\overline{a})^{I^\xi} = \mathbf{f}$ or $\mathbf{u}$ when $\psi_{P_i}(\overline{a})^I$ is $\mathbf{t}$. Denote by $I^{\xi'}$ the interpretation that assigns $P_i(\overline{a})^{I^\xi}$ to $P_i'(\overline{a})$ for every $P_i \in \{P_1, \ldots, P_n\}$ and arbitrary tuple of domain elements $\overline{a}$ and corresponds to I^ξ on all other symbols. There are two observations that can be made about $I^{\xi'}$:

- $I^{\xi'} \leq_p I'$: indeed, $I^\xi \leq_p I$ and for each $P_i' \in \{P_1', \ldots, P_n'\}$ and an arbitrary tuple of domain elements:
  - if $\psi_i(\overline{a})^I = \mathbf{t}$, then $P_i'(\overline{a})^{I'} = \mathbf{f} \geq_p P_i'(\overline{a})^{I^{\xi'}} = P_i(\overline{a})^{I^\xi} = \mathbf{f}$ or $\mathbf{u}$.
  - if $\psi_i(\overline{a})^I = \mathbf{f}$, then $P_i'(\overline{a})^{I'} = P_i(\overline{a})^I \geq_p P_i'(\overline{a})^{I^{\xi'}} = P_i(\overline{a})^{I^\xi}$.
- $\varphi'_{P_j}(\overline{b})^{I^{\xi'}} = \varphi_{P_j}(\overline{b})^{I^\xi} = \mathbf{t}$: obvious from the construction of $I^{\xi'}$ and $\varphi'_{P_j}(\overline{b})$.

Combining these results, we obtain that $\mathbf{t} = \varphi'_{P_j}(\overline{b})^{I^{\xi'}} \leq_p \varphi'_{P_j}(\overline{b})^{I'} = \mathbf{f}$. This is the desired contradiction.

Thus, we obtain that there is no $\psi_i(\overline{x^i})$ and $P_i(\overline{x^i})$ such that $I \models \exists\overline{x^i}(\psi_i(\overline{x^i}) \wedge P_i(\overline{x^i}))$. Consequently, it holds that $\models D, \Gamma, \exists\overline{x^i}(\psi_i(\overline{x}^i) \wedge P_i(\overline{x^i})) \rightarrow \Delta$ for each $i \in \{1, \ldots, n\}$, as required.

Lemma 4.4.6 (Soundness of the definition introduction rule) *Let D be a total definition. If $\models D^R, \Gamma \rightarrow \Delta, \forall\overline{x}(P(\overline{x}) \equiv P^R(\overline{x}))$ for every $P \in Def(D)$, then $\models \Gamma \rightarrow \Delta, D$.*

Proof. Assume that $\models D^R, \Gamma \rightarrow \Delta, \forall\overline{x}(P(\overline{x}) \equiv P^R(\overline{x}))$ for every $P \in Def(D)$ but $\not\models \Gamma \rightarrow \Delta, D$. Then there exists a two-valued Σ-interpretation I such that $I \models \bigwedge \Gamma$ but $I \not\models \bigvee \Delta$, $I \not\models D$. Because D is a total definition, there exists a two-valued Σ-interpretation I' such that I' has the same domain as that of I and I' is the well-founded model of D extending $I|_{Open(D)}$. Because $I \not\models D$, there exists a defined predicate Q of D and some tuple of domain elements $\overline{b}$ such that $Q(\overline{b})^I \neq Q(\overline{b})^{I'}$. Since D is a total definition and D^R is obtained by replacing each occurrence of each defined predicate P in D by P^R, we know that D^R is a total definition. Thus, there exists a two-valued Σ^R-interpretation I'' with the same domain of I such that I'' is the well-founded model of D^R extending I. Notice that for every $P \in Def(D)$ and every tuple of domain elements $\overline{a}$, $P(\overline{a})^{I''} = P(\overline{a})^I$. Because neither Γ nor Δ contains an occurrence of a new predicate P^R, it holds that $I'' \models \bigwedge \Gamma$ and $I'' \not\models \bigvee \Delta$. Therefore, $I'' \models P^R(\overline{a}) \equiv P(\overline{a})$ for every $P \in Def(D)$ and every tuple of domain elements $\overline{a}$. Also, because D^R is obtained by renaming all defined predicates and none of the open predicates and I' and I'' have the same domain, it holds that $P(\overline{a})^{I'} = P^R(\overline{a})^{I''}$ for every $P \in Def(D)$ and every tuple

of domain elements. Hence, $Q(\overline{b})^I = Q(\overline{b})^{I''} = Q^R(\overline{b})^{I''} = Q(\overline{b})^{I'}$, a contradiction. Therefore, $\models \Gamma \to \Delta, D$.

The definition introduction rule is not sound if the inferred definition D is not total. We illustrate it with an example.

Example 4.4.7 *Consider the definition as follows:*

$$D = \{\ \forall x(P(x) \leftarrow \neg P(x))\ \}.$$

Let Γ *and* Δ *be empty sets. It is obvious that* $D^R = \{\ \forall x(P^R(x) \leftarrow \neg P^R(x))\ \}$ *is not total. Thus,* $\models D^R \to \forall x(P^R(x) \equiv P(x))$ *but* $\not\models\!\!\to D$*, which shows that the definition introduction rule is not sound when the inferred definition* D *is non-total.*

It follows that all inference rules in **LFO(ID,SLFP)** except the definition introduction rule are sound with respect to both total and non-total definitions. By induction on the number of inference rules in a proof of a sequent, we obtain the soundness theorem of **LFO(ID,SLFP)**.

Theorem 4.4.8 (Soundness of LFO(ID,SLFP)) *If a sequent* $\Gamma \to \Delta$ *is provable in* **LFO(ID,SLFP)** *without using the definition introduction rule, then* $\models \Gamma \to \Delta$*. If a sequent* $\Gamma \to \Delta$ *is provable in* **LFO(ID,SLFP)** *and all definitions occurring in* Γ *and* Δ *are total, then* $\models \Gamma \to \Delta$*.*

4.5 Completeness of LFO(ID,SLFP)

We now present a proof of the completeness property of **LFO(ID,SLFP)**, namely that the converse of the soundness theorem holds: if $\Gamma \to \Delta$ is valid, then there exists an **LFO(ID,SLFP)** proof of $\Gamma \to \Delta$. However, **LFO(ID,SLFP)** is not complete in general. The main challenge in building a sequent calculus for FO(ID,SLFP) is the representation of unfounded sets. In our approach to this problem, we represent unfounded sets by stratified least fixpoint expressions borrowed from [40]. Compton described stratified least fixpoint logic (SLFP) in [40], which is a logic with a least fixpoint operator and characterizes the expressibility of stratified logic programs. He investigated a deductive inference method for SLFP using the sequent calculus and proved the soundness and completeness of such a proof system. It will be clarified that our proof system **LFO(ID,SLFP)** for FO(ID,SLFP) is complete with respect to a restricted classes of definitions, namely the total $\mathcal{S}$-definitions, the unfounded sets of which can be represented by stratified least fixpoint expressions.

The only inference rules that allows a proof to introduce a positive occurrence of a definition in the succedent of a sequent are the axioms, the weakening rules and the definition introduction rule. As shown in the above section, the definition introduction rule is not sound with respect to non-total definitions. Also, no other inference rule allows a proof to introduce a non-total definition from some first-order formulas. Therefore, one cannot synthesize non-total definitions with **LFO(ID,SLFP)**, i.e., not all valid sequents of the form $\Gamma \rightarrow D$, where D is a non-total definition, can be proved in this deductive system.

We will however prove the completeness property of **LFO(ID,SLFP)** for a fragment of FO(ID,SLFP). First, we define a special class of definitions, $\mathcal{S}$-definitions.

Definition 4.5.1 *A definition D is an* $\mathcal{S}$-definition *if the body of each rule in D satisfies the following conditions:*

- *it is in negation normal form (NNF).*
- *all occurrences of defined predicates in subformulas of the form $\exists x\varphi$ are positive.*
- *all occurrences of defined predicates in subformulas of the form $\forall x\varphi$ are negative.*
- *all occurrences of open predicates are in arbitrary ways.*

Example 4.5.2 *Let* $D_1 =$

$$\left\{ \ \forall x, y(T(x,y) \leftarrow G(x,y) \vee \exists z(T(x,z) \wedge T(z,y))) \ \right\}.$$

It is easy to show that the body $G(x,y) \vee \exists z(T(x,z) \wedge T(z,y))$ is in NNF, and all occurrences the defined predicate T in the existential subformula $\exists z(T(x,z) \wedge T(z,y))$ are positive. So D_1 is an $\mathcal{S}$-definition.

Example 4.5.3 *Let* $D_2 =$

$$\left\{ \begin{array}{c} \forall x, y(T(x,y) \leftarrow G(x,y) \vee \exists z(Q(x,z) \wedge \neg Q(z,y))) \\ \forall x, y(Q(x,y) \leftarrow \neg O(x,y) \vee \forall z(\neg T(x,z) \vee T(z,y))) \end{array} \right\}.$$

D_2 is not an $\mathcal{S}$-definition because there is a positive occurrence of the defined predicate T in the universal subformula $\forall z(\neg T(x,z) \vee T(z,y))$ of the body of the rule defining Q while there is a negative occurrence of the defined predicate Q in the existential subformula $\exists z(Q(x,z) \wedge \neg Q(z,y))$ of the body of the rule defining T.

It is obvious that some familiar types of definitions such as non-recursive definitions and positive definitions [55] are $\mathcal{S}$-definitions. However, not all $\mathcal{S}$-definitions are total. For example, $\{\ \forall x(P(x) \leftarrow \neg P(x))\ \}$ is an $\mathcal{S}$-definition but it is not total. A *total $\mathcal{S}$-definition* is an $\mathcal{S}$-definition that is total.

An *$\mathcal{S}$FO(ID,SLFP) formula* is an FO(ID,SLFP) formula with the restriction that all definitions occurring in it are total $\mathcal{S}$-definitions. $\mathcal{S}$FO(ID,SLFP) is a complete fragment of FO(ID,SLFP) with respect to **LFO(ID,SLFP)**, as shown below.

We now turn to our completeness proof, which employs an extension of the direct style of completeness proof for sequent calculus for standard first-order logic as given in e.g., [32, 178]. The structure of our proof is then roughly as follows:

1. Assume that $\Gamma \to \Delta$ is not provable, and using a *schedule* on which every formula of $\mathcal{S}$FO(ID,SLFP) appears infinitely often, construct from $\Gamma \to \Delta$ a *limit sequent* $\Gamma_\omega \to \Delta_\omega$, where Γ_ω and Δ_ω are infinite sets, such that no finite subsequent of $\Gamma_\omega \to \Delta_\omega$ is provable.
2. Define an equivalence relation $\sim$ on the terms of Σ that essentially "factors out" the equality formulas appearing in Γ_ω and use $\Gamma_\omega \to \Delta_\omega$ to construct a three-valued Σ-structure M_ω interpreting all symbols in Σ.
3. Use M_ω to construct a two-valued Σ-structure W_ω.
4. Prove that $\Gamma_\omega \to \Delta_\omega$ is false in W_ω.
5. It follows from step 3 and 4 that every finite subsequent of $\Gamma_\omega \to \Delta_\omega$ is false in W_ω, including $\Gamma \to \Delta$, so $\Gamma \to \Delta$ is not valid.

The analogous versions of Step 1,2,4,5 also appear in the standard completeness proof for first-order logic (although the equivalence relation $\sim$ constructed in step 2 need only be defined if one considers first-order logic with equality). However, in our setting, the construction of the limit sequent in step 1 must take account of SLFP rules and the definition introduction rule. Applications of the construction of well-founded inductions for definitions must also be accounted for in step 2 and step 3.

Definition 4.5.4 (Schedule) *An* **LFO(ID,SLFP)**-schedule element *of* Σ *is defined as follows:*

- *any definition D is an* **LFO(ID,SLFP)***-schedule element;*
- *any formula of the form $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\varphi \supset \psi$ is an* **LFO(ID,SLFP)***-schedule element;*

- *for any term t of Σ, any variable x and any formula φ, the pairs $\langle \forall x\varphi, t\rangle$ and $\langle \exists x\varphi, t\rangle$ are* **LFO(ID,SLFP)***-schedule elements.*
- *for any formula of the form $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$, the tuple $\langle [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_1\psi,\ldots, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_{m\in\omega}\psi,\ldots\rangle$ is an* **LFO(ID,SLFP)***-schedule element.*

An **LFO(ID,SLFP)**-schedule *for Σ is then an enumeration $(E_i)_{i\leq 0}$ of schedule elements of Σ such that every schedule element of Σ appears infinitely often in the enumeration.*

Note that since we assume that our languages are countable, it follows that the terms of Σ are enumerable. By the same type of argument, the Σ-formulas of FO(ID,SLFP) are enumerable, and hence the **LFO(ID,SLFP)**-schedule elements of Σ are enumerable, so an **LFO(ID,SLFP)**-schedule indeed exists for Σ.

Definition 4.5.5 (Limit sequent) *Suppose that $\Gamma \to \Delta$ is not provable. We define an infinite sequence $(\Gamma_i \to \Delta_i)_{i\geq 0}$ of sequents such that for each $i \geq 0$, $\Gamma_i \to \Delta_i$ is not provable and $\Gamma_i \to \Delta_i$ is a subsequent of $\Gamma_{i+1} \to \Delta_{i+1}$. We set $\Gamma_0 \to \Delta_0 = \Gamma \to \Delta$, so this is trivially true for $i = 0$. We inductively assume we have constructed $(\Gamma_j \to \Delta_j)_{0\leq i\leq j}$ with $\Gamma_i \to \Delta_i$ not provable for all $i \in \{0,\ldots,j\}$, and show how to construct the next sequent $S = (\Gamma_{j+1}, \Delta_{j+1})$.*

We first remark that no formula can be in both Γ_j and Δ_j, otherwise $\Gamma_j \to \Delta_j$ would be provable via one of the axioms of **LFO(ID,SLFP)***. Let E be the $(j+1)$th element in the schedule. We proceed by case distinction on E:*

- *$E = \neg\varphi$. If $\neg\varphi \notin \Gamma_j \cup \Delta_j$ then just define $S = \Gamma_j \to \Delta_j$.*
  If $\neg\varphi \in \Gamma_j$ then consider the rule applications:

$$\dfrac{\dfrac{\Gamma_j \to \varphi, \Delta_j}{\Gamma_j, \neg\varphi \to \Delta_j}\ \textit{left } \neg}{\Gamma_j \to \Delta_j}\ \textit{left contraction}$$

  It is clear that $\Gamma_j \to \varphi, \Delta_j$ is not provable since otherwise $\Gamma_j \to \Delta_j$ would be provable. We define $S = \Gamma_j \to \varphi, \Delta_j$.
  If $\neg\varphi \in \Delta_j$ then consider:

$$\dfrac{\dfrac{\Gamma_j, \varphi \to \Delta_j}{\Gamma_j \to \neg\varphi, \Delta_j}\ \textit{right } \neg}{\Gamma_j \to \Delta_j}\ \textit{right contraction}$$

Again, it is clear that $\Gamma_j, \varphi \to \Delta_j$ *cannot be provable, and we define* $S = \Gamma_j, \varphi \to \Delta_j$.

- $E = \varphi \wedge \psi$. *If* $\varphi \wedge \psi \notin \Gamma_j \cup \Delta_j$ *then just define* $S = \Gamma_j \to \Delta_j$.

  If $\varphi \wedge \psi \in \Gamma_j$ *then consider the rule applications:*

$$\dfrac{\dfrac{\Gamma_j, \varphi, \psi \to \Delta_j}{\Gamma_j, \varphi \wedge \psi \to \Delta_j}\ \textit{left} \wedge}{\Gamma_j \to \Delta_j}\ \textit{left contraction}$$

  It is clear that $\Gamma_j, \varphi, \psi \to \Delta_j$ *is not provable, since otherwise* $\Gamma_j \to \Delta_j$ *would be provable via the rule applications above. We thus define* $S = \Gamma_j, \varphi, \psi \to \Delta_j$.

  On the other hand, if $\varphi \wedge \psi \in \Delta_j$ *then consider:*

$$\dfrac{\dfrac{\Gamma_j \to \varphi, \Delta_j \qquad \Gamma_j \to \psi, \Delta_j}{\Gamma_j \to \varphi \wedge \psi, \Delta_j}\ \textit{right} \wedge}{\Gamma_j \to \Delta_j}\ \textit{right contraction}$$

  From the above it is clear that one of $\Gamma_j \to \Delta_j, \varphi$ *or* $\Gamma_j \to \Delta_j, \psi$ *is not provable. We define* S *to be* $\Gamma_j \to \Delta_j, \varphi$ *if* $\Gamma_j \to \Delta_j, \varphi$ *is not provable and* $\Gamma_j \to \Delta_j, \psi$ *otherwise.*

- $E = \varphi \vee \psi$. *If* $\varphi \vee \psi \notin \Gamma_j \cup \Delta_j$ *then just define* $S = \Gamma_j \to \Delta_j$.

  If $\varphi \vee \psi \in \Gamma_j$ *then by a similar argument to the previous case, one of* $\Gamma_j, \varphi \to \Delta_j$, $\Gamma_j, \psi \to \Delta_j$ *is not provable. We define* S *to be* $\Gamma_j, \varphi \to \Delta_j$ *if this is not provable and* $\Gamma_j, \psi \to \Delta_j$ *otherwise.*

  Similarly, if $\varphi \vee \psi \in \Delta_j$ *then define* $S = \Gamma_j \to \Delta_j, \varphi, \psi$, *which can similarly be seen to be not provable.*

- $E = \varphi \supset \psi$. *If* $\varphi \supset \psi \notin \Gamma_j \cup \Delta_j$ *then just define* $S = \Gamma_j \to \Delta_j$.

  If $\varphi \supset \psi \in \Gamma_j$ *then, by a similar argument to the previous two cases, either* $\Gamma_j, \psi \to \Delta$ *or* $\Gamma_j \to \varphi, \Delta_j$ *is not provable. We define* S *to be* $\Gamma_j, \psi \to \Delta_j$ *if this is not provable and* $\Gamma_j \to \varphi, \Delta_j$ *otherwise.*

  On the other hand, if $\varphi \supset \psi \in \Delta_j$ *then define* $S = \Gamma_j, \varphi \to \psi, \Delta_j$, *which can similarly be seen to be not provable.*

- $E = \langle \exists x \varphi, t \rangle$. *If* $\exists x \varphi \notin \Gamma_j \cup \Delta_j$ *then just define* $S = \Gamma_j \to \Delta_j$.

  If $\exists x \varphi \in \Gamma_j$ *then consider:*

$$\dfrac{\dfrac{\Gamma_j, \varphi[x/z] \to \Delta_j}{\Gamma_j, \exists x \varphi \to \Delta_j}\ \textit{left} \exists}{\Gamma_j \to \Delta_j}\ \textit{left contraction}$$

where $z \notin free(\Gamma_j \cup \Delta_j)$. *It is clear that* $\Gamma_j, \varphi[x/z] \to \Delta_j$ *cannot be provable and we thus define* $S = \Gamma_j, \varphi[x/z] \to \Delta_j$.

On the other hand, if $\exists x\varphi \in \Delta_j$ *then consider:*

$$\dfrac{\dfrac{\Gamma_j \to \varphi[x/t], \Delta_j}{\Gamma_j \to \exists x\varphi, \Delta_j}\ \textit{right } \exists}{\Gamma_j \to \Delta_j}\ \textit{right contraction}$$

It is clear that $\Gamma_j \to \varphi[x/t], \Delta_j$ *cannot be provable and we thus define* $S = \Gamma_j \to \varphi[x/t], \Delta_j$.

- $E = \langle \forall x\varphi, t\rangle$. *If* $\forall x\varphi \notin \Gamma_j \cup \Delta_j$ *then just define* $S = \Gamma_j \to \Delta_j$.
  If $\forall x\varphi \in \Gamma_j$ *then define* $S = \Gamma_j, \varphi[x/t] \to \Delta_j$. *If* $\forall x\varphi \in \Delta_j$ *then define* $S = \Gamma_j \to \varphi[x/z], \Delta_j$, *where* $z \notin free(\Gamma_j \cup \Delta_j)$. *The justification that* S *is not provable is similar to the previous case* $E = \langle \exists x\varphi, t\rangle$.

- $E = \langle [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_1\psi, \ldots, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_{m\in\omega}\psi, \ldots\rangle$. *If* $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \notin \Gamma_j \cup \Delta_j$ *then just define* $S = \Gamma_j \to \Delta_j$.
  If $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \in \Gamma_j$, *then consider the rule applications:*

$$\dfrac{\dfrac{\Gamma_j, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \to \Delta_j\ (m \in \omega)}{\Gamma_j, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \to \Delta_j}\ \textit{left SLFP rule}}{\Gamma_j \to \Delta_j}\ \textit{left contraction}$$

It is clear that for some $m \in \omega$,

$$\Gamma_j, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \to \Delta_j$$

is not provable, since otherwise, for each $m \in \omega$,

$$[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \to \Delta_j$$

is provable , and thus,

$$\Gamma_j, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \to \Delta_j$$

is provable and it follows that $\Gamma_j \to \Delta_j$ *is provable. We thus define* $S =$

$$\Gamma_j, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \to \Delta_j$$

for some $m \in \omega$ *if*

$$\Gamma_j, [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \to \Delta_j$$

is not provable.

If $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi \in \Delta_j$, *then consider the rule applications:*

$$\dfrac{\dfrac{\Gamma_j \to [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi, \Delta_j}{\Gamma_j \to [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi, \Delta_j}\ \textit{right SLFP rule}}{\Gamma_j \to \Delta_j}\ \textit{right contraction}$$

From the above, it is clear that for some fixed m, if $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \notin \Delta_j$ and $\Gamma_j \to [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi, \Delta_j$ is provable, then $\Gamma_j \to \Delta_j$ is provable. Thus, we define $S = \Gamma_j \to [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi, \Delta_j$ where m is the least number such that $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi \notin \Delta_j$ and $\Gamma_j \to [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi, \Delta_j$ is not provable.

- *$E = D$. Suppose that $P_1,\ldots,P_n$ are all defined predicates of D. If $D \notin \Gamma_j \cup \Delta_j$ then just define $S = \Gamma_j \to \Delta_j$.*

  If $D \in \Gamma_j$ then just define $S = \Gamma_j \to \Delta_j$.

  If $D \in \Delta_j$ then consider:

$$\dfrac{\dfrac{\begin{array}{c}\Gamma_j, D^R \to \Delta_j, \forall\overline{x^1}(P_1^R(\overline{x^1}) \equiv P_1(\overline{x^1}))\\ \vdots \\ \Gamma_j, D^R \to \Delta_j, \forall\overline{x^n}(P_n^R(\overline{x^n}) \equiv P_n(\overline{x^n}))\end{array}}{\Gamma_j \to \Delta_j, D}\ \textit{definition introduction}}{\Gamma_j \to \Delta_j}\ \textit{right contraction}$$

  It is clear that $\Gamma_j, D^R \to \Delta_j, \forall\overline{x^i}(P_i^R(\overline{x^i}) \equiv P_1(\overline{x^i}))$ cannot be provable for some $i \in \{1,\ldots,n\}$. We thus define $S = \Gamma_j, D^R \to \Delta_j, \forall\overline{x^i}(P_i^R(\overline{x^i}) \equiv P_1(\overline{x^i}))$ where i is the least number in $\{1,\ldots,n\}$ such that $\Gamma_j, D^R \to \Delta_j, \forall\overline{x^i}(P_i^R(\overline{x^i}) \equiv P_1(\overline{x^i}))$ is not provable.

Observe that by construction, we have $\Gamma_j \subseteq \Gamma_{j+1}$ and $\Delta_j \subseteq \Delta_{j+1}$ for $j \geq 0$. Let $\Gamma_\omega = \bigcup_{j\geq 0}\Gamma_j$ and $\Delta_\omega = \bigcup_{j\geq 0}\Delta_j$. Then the limit sequent for $\Gamma \to \Delta$ *is defined to be $\Gamma_\omega \to \Delta_\omega$. Of course, the limit sequent is not strictly speaking a sequent, since in general Γ_ω and Δ_ω will be infinite sets. When we say that e.g., $\Gamma_\omega \to \Delta_\omega$ is provable, we mean that $\Gamma' \to \Delta'$ is provable for some finite $\Gamma' \subseteq \Gamma_\omega$ and $\Delta' \subseteq \Delta_\omega$.*

We remark that the rules for equality, the right definition rule and the left definition rule are not used when constructing the limit sequent in the definition above. The roles of equality and of introduction of defined predicates are instead accounted for by separate lemmas that appear below.

For the remainder of this section, until the statement of our completeness theorem, we shall assume that some fixed sequent $\Gamma \to \Delta$ is not provable, and that $\Gamma_\omega \to \Delta_\omega$ is the limit sequent for $\Gamma \to \Delta$ constructed according to Definition 4.5.5.

Definition 4.5.6 *Define the relation $\sim$ on terms of Σ by:*

$$\frac{t_1 = t_2 \in \Gamma_\omega}{t_1 \sim t_2}\ (\sim Base) \quad \frac{}{t \sim t}\ (\sim Refl) \quad \frac{t_1 \sim t_2}{t_2 \sim t_1}\ (\sim Sym)$$

$$\frac{t_1 \sim t_2\ t_2 \sim t_3}{t_1 \sim t_3}\ (\sim Trans) \quad \frac{t_1 \sim u_1 \ldots t_k \sim u_k}{f(t_1, \ldots, t_k) \sim f(u_1, \ldots, u_k)}\ (\sim Cong)$$

where f is an arbitrary function symbol of arity k in Σ in the rule $\sim$Cong. We call $\sim$ the equivalence relation induced by $\Gamma_\omega \to \Delta_\omega$, and write $[t]$ for the equivalence class of t with respect to $\sim$, i.e., $[t] = \{u \mid t \sim u\}$.

A proof system **LFO(SLFP)** for FO(SLFP) is obtained from **LFO(ID,SLFP)** by eliminating definition rules. It is not difficult to prove the completeness of **LFO(SLFP)**, but we spell out the details of the completeness proof of **LFO(SLFP)**, namely that if $\models \Gamma \to \Delta$, then $\Gamma \to \Delta$ is provable in **LFO(SLFP)**, where Γ and Δ are sets of FO(SLFP) formulas, for the sake of completeness. The completeness property of **LFO(SLFP)** plays a key role in the completeness proof for **LFO(ID,SLFP)**, as shown later.

Definition 4.5.7 (Counter-interpretation) *Define a Σ-interpretation M_ω by:*

- *the domain of M_ω is $Terms(\Sigma)/\sim$, the set of $\sim$-equivalence classes of Σ-terms;*
- *for any constant symbol c, $c^{M_\omega} = [c]$;*
- *for any variable symbol x, $x^{M_\omega} = [x]$;*
- *for any function symbol f in Σ of arity k, $f^{M_\omega}([t_1], \ldots, [t_k]) = [f(t_1, \ldots, t_k)]$;*
- *for any predicate symbol P in Σ of arity k, $P^{M_\omega}([t_1], \ldots, [t_k]) = \mathbf{t} \Leftrightarrow \exists u_1, \ldots, u_k . t_1 \sim u_1, \ldots, t_k \sim u_k$ and $P(u_1, \ldots, u_k) \in \Gamma_\omega$.*

M_ω is called the counter-interpretation *for $\Gamma_\omega \to \Delta_\omega$.*

Proposition 4.5.8 *For all terms of Σ we have $t^{M_\omega} = [t]$.*

Proof. By induction on the structure of t.

- **Case** $t = x$. We have $x^{M_\omega} = [x]$ by definition of M_ω.
- **Case** $t = c$. We have $c^{M_\omega} = [c]$ by definition of M_ω.

- **Case** $t = f(t_1, \ldots, t_k)$. We have:

$$\begin{array}{ll} f(t_1, \ldots, t_k)^{M_\omega} & = f^{M_\omega}([t_1], \ldots, [t_k]) \quad \text{by the induction hypothesis} \\ & = [f(t_1, \ldots, t_k)] \quad \text{by definition of } M_\omega \end{array}$$

Proposition 4.5.9 *If $t(x_1, \ldots, x_k)$ is a term (whose variable are all contained in $\{x_1, \ldots, x_k\}$), and $t_1, \ldots, t_k$ are terms, we have $[t(t_1, \ldots, t_k)] = t^{M_\omega}([t_1], \ldots, [t_k])$.*

Proof. By induction on the structure of t.

- **Case** $t = x$. As the variables of t are all contained in $\{x_1, \ldots, x_k\}$, we have $t = x_i$ for some $i \in \{1, \ldots, k\}$. We therefore require to prove $[x_i] = [x_i]$, which is trivial.
- **Case** $t = c$. We are required to prove $[c] = c^{M_\omega}$, which is immediately the case by definition of M_ω.
- **Case** $t = f(u_1, \ldots, u_m)$. By assumption we have $Var(u_i) \subseteq \{x_1, \ldots, x_k\}$ and so we can write u_i as $u_i(x_1, \ldots, x_k)$ for all $i \in \{1, \ldots, m\}$. Then we have:

$$\begin{array}{ll} [(f(u_1, \ldots, u_m))(t_1, \ldots, t_k)] & = [f(u_1(t_1, \ldots, t_k), \ldots, u_m(t_1, \ldots, t_k))] \\ & = f^{M_\omega}([u_1(t_1, \ldots, t_k)], \ldots, [u_m(t_1, \ldots, t_k)]) \\ & = f^{M_\omega}(u_1^{M_\omega}([t_1], \ldots, [t_k]), \ldots u_m^{M_\omega}([t_1], \ldots, [t_k])) \\ & = (f(u_1, \ldots, u_m))^{M_\omega}([t_1], \ldots, [t_k]) \end{array}$$

Lemma 4.5.10 *If $t \sim u$ then, for any formula φ, it holds that $\Gamma_\omega \to \varphi[x/t]$ is provable iff $\Gamma_\omega \to \varphi[x/u]$ is provable.*

Proof. By rule induction on $t \sim u$.

Case ($\sim$ Base). We have $t = u \in \Gamma_\omega$. Suppose $\Gamma_\omega \to \varphi[x/t]$ is provable, i.e., $\Gamma' \to \varphi[x/t]$ is provable for some finite $\Gamma' \subseteq \Gamma_\omega$. Without loss of generality, we can assume $t = u \in \Gamma'$ (for if not, we can weaken the derivation of $\Gamma' \to \varphi[x/t]$ everywhere by $t = u$ on the left, possibly renaming some eigenvariables). Then we can derive $\Gamma' \to \varphi[x/u]$ as follows:

$$\dfrac{\dfrac{\Gamma' \to \varphi[x/t]}{\Gamma', t = u \to \varphi[x/u]}\,\text{left equality}}{\Gamma' \to \varphi[x/u]}\,\text{left contraction}$$

hence it follows that $\Gamma_\omega \to \varphi[x/u]$ is provable because $\Gamma' \subseteq \Gamma_\omega$. The converse is similar.

Case ($\sim$ Refl). We require to prove that $\Gamma_\omega \to \varphi[x/t]$ is provable iff $\Gamma_\omega \to \varphi[x/t]$ is provable, which is trivially the case.

Case ($\sim$ Sym). By induction hypothesis, we have that $\Gamma_\omega \to \varphi[x/t_1]$ is provable iff $\Gamma_\omega \to \varphi[x/t_2]$ is provable. It immediately follows that $\Gamma_\omega \to \varphi[x/t_2]$ is provable iff $\Gamma_\omega \to \varphi[x/t_1]$ is provable.

Case ($\sim$ Trans). By induction hypothesis, we have that $\Gamma_\omega \to \varphi[x/t_1]$ is provable iff $\Gamma_\omega \to \varphi[x/t_2]$ is provable iff $\Gamma_\omega \to \varphi[x/t_3]$ is provable. It immediately follows that $\Gamma_\omega \to \varphi[x/t_1]$ is provable iff $\Gamma_\omega \to \varphi[x/t_3]$ is provable.

Case ($\sim$ Cong). By induction hypothesis we have that for all formulas φ and all $i \in \{1, \ldots, k\}$, $\Gamma_\omega \to \varphi[x/t_i]$ is provable iff $\Gamma_\omega \to \varphi[x/u_i]$ is provable. We require to show that $\Gamma_\omega \to \varphi[x/f(t_1, \ldots, t_k)]$ is provable iff $\Gamma_\omega \to \varphi[x/f(u_1, \ldots, u_k)]$ is provable. By successively applying the induction hypotheses, we obtain:

$$\begin{array}{ll} & \Gamma_\omega \to \varphi[x/f(t_1, \ldots, t_k)] \text{ is provable} \\ \Leftrightarrow & \Gamma_\omega \to \varphi[x/f(u_1, t_2, t_3, \ldots, t_k)] \text{ is provable} \\ \Leftrightarrow & \Gamma_\omega \to \varphi[x/f(u_1, u_2, t_3, t_4, \ldots, t_k)] \text{ is provable} \\ \vdots & \\ \Leftrightarrow & \Gamma_\omega \to \varphi[x/f(u_1, \ldots, u_k)] \text{ is provable} \end{array}$$

which completes the case.

Lemma 4.5.11 *If $\varphi \in \Gamma_\omega$ then $M_\omega \models \varphi$, and if $\varphi \in \Delta_\omega$ then $M_\omega \not\models \varphi$.*

Proof. By induction on the structure of φ.

- **Case** $\varphi = P(t_1, \ldots, t_k)$. Suppose $P(t_1, \ldots, t_k) \in \Gamma_\omega$. As $t_i \sim t_i$ for all $i \in \{1, \ldots, k\}$ by ($\sim$Refl), we thus have $P^{M_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ by definition of M_ω, i.e., $M_\omega \models P(t_1, \ldots, t_k)$, as desired.

  Now suppose $P(t_1, \ldots, t_k) \in \Delta_\omega$ and suppose for contradiction that $M_\omega \models P(t_1, \ldots, t_k)$, i,e., $P^{M_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ holds. Then there exist terms $u_1, \ldots, u_k$ such that $t_1 \sim u_1, \ldots, t_k \sim u_k$ and $P(u_1, \ldots, u_k) \in \Gamma_\omega$. So there is a proof of $\Gamma_\omega \to P(u_1, \ldots, u_k)$ (via an application of axiom). By Lemma 4.5.10 (applied k times), it follows that there is a proof of $\Gamma_\omega \to P(t_1, \ldots, t_k)$. Since $P(t_1, \ldots, t_k) \in \Delta_\omega$, there is a proof of $\Gamma_\omega \to \Delta_\omega$, which is a contradiction. Hence $M_\omega \not\models P(t_1, \ldots, t_k)$, as desired.

- **Case** $\varphi = t_1 = t_2$. If $t_1 = t_2 \in \Gamma_\omega$ then we have $t_1 \sim t_2$ (by ($\sim$Base)) and thus $[t_1] = [t_2]$. By Proposition 4.5.8 it follows that ${t_1}^{M_\omega} = {t_2}^{M_\omega}$ and so $M_\omega \models t_1 = t_2$.

Now suppose that $t_1 = t_2 \in \Delta_\omega$, and suppose for contradiction that $M_\omega \models t_1 = t_2$, i.e., ${t_1}^{M_\omega} = {t_2}^{M_\omega}$. By Proposition 4.5.8, $[t_1] = [t_2]$ and so $t_1 \sim t_2$. Now observe that $\Gamma_\omega \to t_1 = t_1$ is provable via an application of axiom. Hence by Lemma 4.5.10, $\Gamma_\omega \to t_1 = t_2$ is also provable. But since $t_1 = t_2 \in \Delta_\omega$, we would then have a proof of $\Gamma_\omega \to \Delta_\omega$, which is an contradiction. Hence $M_\omega \not\models t_1 = t_2$.

- **Case** $\varphi = \neg\varphi'$. If $\neg\varphi' \in \Gamma_\omega$, then by the construction of $\Gamma_\omega \to \Delta_\omega$ (c.f. Definition 4.5.5), there is an $i \geq 0$ such that for all $j \geq i$, $\neg\varphi' \in \Gamma_j$. Furthermore, it follows that $\varphi' \in \Delta_j$ for some $j \geq i$ and hence $\varphi' \in \Delta_\omega$. By induction hypothesis, we thus have $M_\omega \not\models \varphi'$, i.e., $M_\omega \models \neg\varphi$, as desired.

  Now suppose $\neg\varphi' \in \Delta_\omega$. By construction of $\Gamma_\omega \to \Delta_\omega$, $\varphi' \in \Gamma_\omega$ so by the induction hypothesis we have $M_\omega \models \varphi'$, i.e., $M_\omega \not\models \neg\varphi'$, as desired.

- **Case** $\varphi = \varphi_1 \wedge \varphi_2$. If $\varphi_1 \wedge \varphi_2 \in \Gamma_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, $\varphi_1 \in \Gamma_\omega$ and $\varphi_2 \in \Gamma_\omega$. By induction hypothesis $M_\omega \models \varphi_1$ and $M_\omega \models \varphi_2$, i.e., $M_\omega \models \varphi_1 \wedge \varphi_2$, as desired.

  Now if $\varphi_1 \wedge \varphi_2 \in \Delta_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$ we have $\varphi_1 \in \Delta_\omega$ or $\varphi_2 \in \Delta_\omega$. In the former case we have by the induction hypothesis $M_\omega \not\models \varphi_1$ whence it is clear that $M_\omega \not\models \varphi_1 \wedge \varphi_2$; the other case is similar.

- **Case** $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \supset \varphi_2$. These cases are similar to the previous case $\varphi = \varphi_1 \wedge \varphi_2$ above.

- **Case** $\varphi = \exists x\varphi'$. If $\exists x\varphi' \in \Gamma_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, we have $\varphi'[x/z] \in \Gamma_\omega$ for some variable z, whence $M_\omega \models \varphi'[x/z]$ by the induction hypothesis. It follows directly that $M_\omega \models \exists x\varphi'$.

  Now suppose $\exists x\varphi' \in \Delta_\omega$, and observe that by construction of $\Gamma_\omega \to \Delta_\omega$ there is an $i \geq 0$ such that $\exists x\varphi' \in \Delta_j$ for all $j \geq i$. Now consider an arbitrary term t of Σ and note that the element $\langle \exists x\varphi', t\rangle$ appears infinitely often on the schedule $(E_i)_{i\geq 0}$ according to which $\Gamma_\omega \to \Delta_\omega$ is constructed. So there is a $j \geq i$ such that $E_j = \langle \exists x\varphi', t\rangle$ and thus we have $\varphi'[x/t] \in \Delta_\omega$. As t was chosen arbitrarily, it follows that for every term t, $\varphi'[x/t] \in \Delta_\omega$. So by the induction hypothesis $M_\omega \not\models \varphi'[x/t]$ for every term t. Suppose for contradiction that $M_\omega \models \exists x\varphi'$. Then for some $t \in Term(\Sigma)$, we would have $M_\omega \models \varphi'[x/t]$, which contradicts our induction hypotheses. Hence $M_\omega \not\models \exists x\varphi'$.

- **Case** $\varphi = \forall x\varphi'$. This case is similar to the case $\varphi = \exists x\varphi'$ above.

- **Case** $\varphi = [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$.
  If $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k) \in \Gamma_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k) \in \Gamma_\omega$ for some m. By the induction hypothesis, we have

$M_\omega \models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k)$ for some m and hence, it holds that $M_\omega \models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k)$, as desired.

Now if $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k) \in \Delta_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k) \in \Delta_\omega$ for all m. By the induction hypothesis, we have $M_\omega \not\models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k)$ for all m and hence, it holds that $M_\omega \not\models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k)$, as desired.

Now we state the completeness theorem for **LFO(SLFP)**.

Theorem 4.5.12 (Completeness theorem of LFO(SLFP)) *Let Γ and Δ be arbitrary sets of FO(SLFP) formulas. If $\Gamma \to \Delta$ is valid, then it is provable in* **LFO(SLFP)**.

Proof. Suppose for contradiction that $\Gamma \to \Delta$ is valid, but not provable in **LFO(SLFP)**, and let $\Gamma_\omega \to \Delta_\omega$ be the limit sequent for $\Gamma \to \Delta$ (c.f. Definition 4.5.5) with counter-interpretation M_ω (c.f. Definition 4.5.7). By Lemma 4.5.11, we have $M_\omega \models \varphi$ for all $\varphi \in \Gamma_\omega$ and $M_\omega \not\models \varphi$ for all $\varphi \in \Delta_\omega$. As $\Gamma \subseteq \Gamma_\omega$ and $\Delta \subseteq \Delta_\omega$, we thus have $M_\omega \models \varphi$ for all $\varphi \in \Gamma$ and $M_\omega \not\models \varphi$ for all $\varphi \in \Delta$.

In other words, we have constructed a Σ-interpretation M_ω such that $\Gamma \to \Delta$ is not true in M_ω and we have the required contradiction. Hence $\Gamma \to \Delta$ is provable in **LFO(SLFP)**.

We now turn to the proof of the completeness property of **LFO(ID,SLFP)**, namely that if $\Gamma \to \Delta$ is valid, where Γ and Δ are arbitrary sets of $\mathcal{S}$FO(ID,SLFP) formulas, then $\Gamma \to \Delta$ is provable in **LFO(ID,SLFP)**.

Given a definition D over Σ and a three-valued Σ-interpretation I, the set of all unfounded sets of D in I is denoted by $U_D(I)$. The *maximal unfounded set* of D in I is defined by $\bigcup_{X \in U_D(I)} X$. The maximal unfounded set of an $\mathcal{S}$-definition can be represented by the negation of stratified least fixpoint expressions, which is an important property of $\mathcal{S}$-definitions and will be applied in the completeness proof for **LFO(ID,SLFP)**. We demonstrate this property as follows.

Definition 4.5.13 *Let Σ be a vocabulary, D a definition over Σ and I a three-valued Σ-interpretation. Let X be a set of defined domain atoms of I. Define $\mathcal{R}_{D,I} : X \to X$ by $\mathcal{R}_{D,I}(X) = \{P(\overline{a}) \in X \mid \varphi_P(\overline{a})^{I[X/\mathbf{f}]} = \mathbf{f}\}$.*

It is rather easy to see that $\mathcal{R}_{D,I}$ is a monotonic operator. Given a set X of defined domain atoms of I, it is obvious that the sequence:

$$R_0 = X$$
$$R_n = \mathcal{R}_{D,I}(R_{n-1}) \quad (n \geq 1)$$

decreases monotonically and converges to a limit, denoted by $\mathcal{R}^{\alpha}_{D,I}(X)$.

Lemma 4.5.14 *Let D be a definition and $(I^{\xi})_{\xi\leq\alpha}$ a well-founded induction of D. Let $I^{\xi+1} = I^{\xi}[U/\mathbf{f}]$ where U is the maximal unfounded set of D in I^{ξ}. Suppose that $\{P_1, \ldots, P_m\}$ is the set of all defined predicates occurring in U and $F = \{P_i(\overline{a}) \mid P_i \in \{P_1, \ldots, P_m\}$ and $\overline{a}$ is an arbitrary tuple of domain elements$\}$. Then $U = \mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$.*

Proof. Firstly, we want to show that $\mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$ is an unfounded set of D in I^{ξ}. According to the construction of $\mathcal{R}^{\alpha}_{D,I^{\xi}}$, it is obvious that for each defined domain atom $P(\overline{a}) \in \mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$, it holds that $\varphi_P(\overline{a})^{I^{\xi}[\mathcal{R}^{\alpha}_{D,I^{\xi}}(F)/\mathbf{f}]} = \mathbf{f}$. Thus, $\mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$ is an unfounded set of D in I^{ξ}. On the other hand, we want to show that $U \subseteq \mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$. Because U is an unfounded set of D in I^{ξ}, it is obvious that $U = \mathcal{R}_{D,I^{\xi}}(U)$ and moreover, $U = \mathcal{R}^{\alpha}_{D,I^{\xi}}(U)$. It is clear that $U \subseteq F$. Then by the monotonicity of the operator $\mathcal{R}_{D,I^{\xi}}$, we have that $U = \mathcal{R}^{\alpha}_{D,I^{\xi}}(U) \subseteq \mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$. Therefore, it holds that $U = \mathcal{R}^{\alpha}_{D,I^{\xi}}(F)$.

By the completeness of FO(SLFP) as shown by Theorem 4.5.12 and Lemma 4.5.14, we can easily obtain the following property.

Proposition 4.5.15 *Let D be an $\mathcal{S}$-definition and $(I^{\xi})_{\xi\leq\alpha}$ a well-founded induction of D. Let $I^{\xi+1} = I^{\xi}[U/\mathbf{f}]$ where U is the maximal unfounded set of D in I^{ξ}. Suppose that $\{P_1, \ldots, P_m\}$ is the set of all defined predicates occurring in U. Then $\bigcup_{i\in\{1,\ldots m\}}\{P_i(\overline{x^i}) \mid \neg([LFP_{X_1(\overline{x^1}),\ldots,X_m(\overline{x^m})}(\varphi_{P_1}(P_1/X_1, \ldots, P_m/X_m), \ldots, \varphi_{P_m}(P_1/X_1, \ldots, P_m/X_m))]X_i(\overline{x^i}))\}$ is a symbolic representation of U.*

Definition 4.5.16 (A three-valued counter-interpretation) *Define a three-valued interpretation M_{ω} for Σ by:*

- *the domain of M_{ω} is $Terms(\Sigma)/\sim$, the set of $\sim$-equivalence classes of Σ-terms;*
- *for any constant symbol c, $c^{M_{\omega}} = [c]$;*
- *for any variable symbol x, $x^{M_{\omega}} = [x]$;*
- *for any function symbol f in Σ of arity k, $f^{M_{\omega}}([t_1], \ldots, [t_k]) = [f(t_1, \ldots, t_k)]$;*
- *for any open predicate symbol O in Σ of arity k,*
  - *$O^{M_{\omega}}([t_1], \ldots, [t_k]) = \mathbf{t} \Leftrightarrow \exists u_1, \ldots, u_k.t_1 \sim u_1, \ldots, t_k \sim u_k$ and $O(u_1, \ldots, u_k) \in \Gamma_{\omega}$;*

– $O^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{f}$, *otherwise;*

- *for any definition D occurring in Γ_ω, defining a well-founded induction of D extending $M_\omega|_{Open(D)}$ as a sequence $(I^\xi)_{\xi\leq\alpha}$ of three-valued Σ-interpretation such that:*
  - *for every defined predicate symbol P, $P^{I^0} = \mathbf{u}$,*
  - *for each limit ordinal $\lambda \leq \alpha$, $I^\lambda = lub_{\leq_p}(\{I^\xi \mid \xi < \lambda\})$, and*
  - *for every ordinal ξ, $I^{\xi+1}$ relates to I^ξ in one of the following ways:*
    - *$I^{\xi+1} = I^\xi[P(t_1,\ldots,t_k)/\mathbf{t}]$ for some $P(t_1,\ldots,t_k)$ such that $P(t_1,\ldots,t_k)$ is unknown in I^ξ and $\varphi_P^{I^\xi}(t_1,\ldots,t_k) = \mathbf{t}$;*
    - *$I^{\xi+1} = I^\xi[U/\mathbf{f}]$, where U is the maximal unfounded set of D in I^ξ.*
  - *for any defined predicate symbol P of D in Σ of arity k,*
    - *$P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{t} \Leftrightarrow \exists u_1,\ldots,u_k.t_1 \sim u_1,\ldots,t_k \sim u_k$ and $P^{I^\alpha}(u_1,\ldots,u_k) = \mathbf{t}$;*
    - *$P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{f} \Leftrightarrow \exists u_1,\ldots,u_k.t_1 \sim u_1,\ldots,t_k \sim u_k$ and $P^{I^\alpha}(u_1,\ldots,u_k) = \mathbf{f}$;*
    - *$P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{u}$, otherwise;*
  - *for any other defined predicate symbol P in Σ of arity k that is not defined in any definition D occurring in Γ_ω,*
    - *$P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{t} \Leftrightarrow \exists u_1,\ldots,u_k.t_1 \sim u_1,\ldots,t_k \sim u_k$ and $P(u_1,\ldots,u_k) \in \Gamma_\omega$;*
    - *$P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{f}$, otherwise;*

Lemma 4.5.17 *Let D be an arbitrary definition occurring in Γ_ω. Let Γ' be the set of all open atoms occurring in Γ_ω and let Δ' be the set of all open atoms occurring in Δ_ω. For any defined predicate $P \in Def(D)$, if $P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{t}$, then*

$$\Gamma', D, \neg\Delta' \rightarrow P(t_1,\ldots,t_k)$$

is provable in **LFO(ID,SLFP)**, *and if $P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{f}$, then*

$$\Gamma', D, \neg\Delta' \rightarrow \neg P(t_1,\ldots,t_k)$$

is provable in **LFO(ID,SLFP)**.

Proof. Let $(I^\xi)_{\xi\leq\alpha}$ be a terminal well-founded induction for D extending the interpretation $M_\omega|_{Open(D)}$ for open atoms by:

- for every defined predicate symbol P, $P^{I^0} = \mathbf{u}$,
- for each limit ordinal $\lambda \leq \alpha$, $I^\lambda = lub_{\leq_p}(\{I^\xi \mid \xi < \lambda\})$, and

- for every ordinal ξ, $I^{\xi+1}$ relates to I^{ξ} in one of the following ways:
  - $I^{\xi+1} = I^{\xi}[P(t_1,\ldots,t_k)/\mathbf{t}]$ for some $P(t_1,\ldots,t_k)$ such that $P(t_1,\ldots,t_k)$ is unknown in I^{ξ} and $\varphi_P^{I^{\xi}}(t_1,\ldots,t_k) = \mathbf{t}$;
  - $I^{\xi+1} = I^{\xi}[U/\mathbf{f}]$, where U is the maximal unfounded set of D in I^{ξ}.

Denote by I the limit I^{α}. Denote by Π^{ξ} a set of all defined literals L such that $L^{I^{\xi}} = \mathbf{t}$ in arbitrary order. We want to prove that $\Pi^{\xi}, \Gamma', D, \neg\Delta' \to L$ is provable in **LFO(ID,SLFP)** for all $L \in \Pi^{\xi+1} \setminus \Pi^{\xi}$. For each $L \in \Pi^{\xi+1} \setminus \Pi^{\xi}, L^{I^{\xi}} = \mathbf{u}$ and $L^{I^{\xi+1}} = \mathbf{t}$. We distinguish between the case where $\Pi^{\xi+1} \setminus \Pi^{\xi}$ contains one positive literal and the case where it contains a set of negative literals.

- Assume that $\Pi^{\xi+1} \setminus \Pi^{\xi}$ consists of one defined atom $P(u_1,\ldots,u_k)$. For every two-valued Σ-interpretation J such that J is a model of $\bigwedge \Pi^{\xi}, \bigwedge \Gamma'$ and $\bigwedge \neg\Delta'$, it holds that $I^{\xi} \leq_p J$. $P^{I^{\xi+1}}(u_1,\ldots,u_k) = \mathbf{t}$, hence $\varphi_P^{I^{\xi}}(u_1,\ldots,u_k) = \mathbf{t}$. It follows that $\varphi_P^{J}(u_1,\ldots,u_k) = \mathbf{t}$. Thus, it holds that $\models \Pi^{\xi}, \Gamma', \neg\Delta' \to \varphi_P(u_1,\ldots,u_k)$. Therefore, by the completeness of **LFO(SLFP)**, the sequent $\Pi^{\xi}, \Gamma', \neg\Delta' \to \varphi_P(u_1,\ldots,u_k)$ is provable in **LFO(SLFP)**, and hence, it is provable in **LFO(ID,SLFP)**. Moreover, by the right definition rule, $\Pi^{\xi}, \Gamma', D, \neg\Delta' \to P(u_1,\ldots,u_k)$ is provable in **LFO(ID,SLFP)**.
- For the other case, assume that $\Pi^{\xi+1} \setminus \Pi^{\xi}$ is a set of negative literals. Denote the set $\{P(u_1,\ldots,u_k) \mid \neg P(u_1,\ldots,u_k) \in \Pi^{\xi+1} \setminus \Pi^{\xi}\}$ by U. Suppose that $\{P_1,\ldots,P_m\}$ is the set of all defined predicates occurring in U. According to the construction of the terminal well-founded induction $(I^{\xi})_{\xi\leq\alpha}$, we have that U is the maximal unfounded set of D in I^{ξ}. By Proposition 4.5.15, we know that U can be represented by $\bigcup_{i\in\{1,\ldots m\}}\{P_i(\overline{x^i}) \mid \neg([LFP_{X_1(\overline{x^1}),\ldots,X_m(\overline{x^m})}(\varphi_{P_1}(P_1/X_1,\ldots,P_m/X_m),\ldots,$ $\varphi_{P_m}(P_1/X_1,\ldots,P_m/X_m))]X_i(\overline{x^i}))\}$. Recall that $I^{\xi+1} = I^{\xi}[U/\mathbf{f}]$. $P^{I^{\xi+1}}(u_1,\ldots,u_k) = \mathbf{f}$ for each $P(u_1,\ldots,u_k) \in U$, hence $\varphi_P^{I^{\xi+1}}(u_1,\ldots,u_k) = \mathbf{f}$. Consider the interpretation $I'^{\xi+1} = I^{\xi}[U'/\mathbf{f}]$, where $U' = \{P'(u_1,\ldots,u_k) \mid P(u_1,\ldots,u_k) \in U\}$. There are two simple observations that can be made about $I'^{\xi+1}$ and each $\varphi'_P(u_1,\ldots,u_k)$:
  - $I'^{\xi+1} \leq_p J'$ for every two-valued $\Sigma\cup\{P'_1,\ldots,P'_m\}$-interpretation J' such that J' satisfies $\bigwedge \Pi^{\xi}, \bigwedge \Gamma', \bigwedge \neg\Delta'$ and $\bigwedge \neg U'$, where U' is represented by $\bigcup_{i\in\{1,\ldots m\}}\{P'_i(\overline{x^i}) \mid \neg([LFP_{X_1(\overline{x^1}),\ldots,X_m(\overline{x^m})}(\varphi_{P_1}(P_1/X_1,\ldots,P_m/X_m),\ldots,$ $\varphi_{P_m}(P_1/X_1,\ldots,P_m/X_m))]X_i(\overline{x^i}))\}$.
  - ${\varphi'_P}^{I'^{\xi+1}}(u_1,\ldots,u_k) = {\varphi_P}^{I'^{\xi+1}}(u_1,\ldots,u_k) = \mathbf{f}$: obvious from the construction of $I'^{\xi+1}$ and $\varphi'_P(u_1,\ldots,u_k)$.

Combining these results, we obtain ${\varphi'_P}^{J'}(u_1,\ldots,u_k) = \mathbf{f}$ for every two-valued interpretation J' satisfying $\bigwedge \Pi^\xi, \bigwedge \Gamma', \bigwedge \neg\Delta'$ and $\bigwedge \neg U'$. It follows that $\models \Pi^\xi, \Gamma', \neg\Delta', \forall\overline{x^1}(\neg([LFP_{X_1(\overline{x^1}),\ldots,X_m(\overline{x^m})}(\varphi_{P_1}(P_1/X_1,\ldots,P_m/X_m),\ldots,$
$\varphi_{P_m}(P_1/X_1,\ldots,P_m/X_m))]X_1(\overline{x^1})) \supset \neg P'_1(\overline{x^1})),\ldots,$
$\forall\overline{x^m}(\neg([LFP_{X_1(\overline{x^1}),\ldots,X_m(\overline{x^m})}(\varphi_{P_1}(P_1/X_1,\ldots,P_m/X_m),\ldots,$
$\varphi_{P_m}(P_1/X_1,\ldots,P_m/X_m))]X_m(\overline{x^m})) \supset \neg P'_m(\overline{x^m})) \rightarrow$
$\forall\overline{x^i}(\neg([LFP_{X_1(\overline{x^1}),\ldots,X_m(\overline{x^m})}(\varphi_{P_1}(P_1/X_1,\ldots,P_m/X_m),\ldots,$
$\varphi_{P_m}(P_1/X_1,\ldots,P_m/X_m))]X_m(\overline{x^m})) \supset \neg\varphi'_{P_i}(\overline{x^i}))$ for every $i \in \{1,\ldots,m\}$. By the completeness of **LFO(SLFP)**, the left definition rule, the right $\neg$ rule and the right $\vee$ rule, the sequent $\Pi^\xi, \Gamma', D, \neg\Delta' \rightarrow \neg P(u_1,\ldots,u_k)$ is provable in **LFO(ID,SLFP)** for each $P(u_1,\ldots,u_k) \in U$.

Since $(I^\xi)_{\xi\leq\alpha}$ is a terminal well-founded induction for D with the limit $I = I^\alpha$, it is obvious that the set of defined literals L for which $L^I = \mathbf{t}$ is exactly the set of all defined literals in Π^α. Thus, by using the cut rule, it is easy to show by induction on ξ that if L is a defined literal of D such that $L^I = \mathbf{t}$, the sequent $\Gamma', D, \neg\Delta' \rightarrow L$ is provable in **LPC(ID)**. Thus, we have that for any defined predicate $P \in Def(D)$, if $P^I(u_1,\ldots,u_k) = \mathbf{t}$, then $\Gamma', D, \neg\Delta' \rightarrow P(u_1,\ldots,u_k)$ is provable in **LFO(ID,SLFP)**, and if $P^I(u_1,\ldots,u_k) = \mathbf{f}$, then $\Gamma', D, \neg\Delta' \rightarrow \neg P(u_1,\ldots,u_k)$ is provable in **LFO(ID,SLFP)**. For any defined predicate $P \in Def(D)$, if $P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{t}$, then by definition of M_ω (c.f. Definition 4.5.16) we have that there exist $u_1,\ldots,u_k$ such that $t_1 \sim u_1,\ldots,t_k \sim u_k$ and $P^I(u_1,\ldots,u_k) = \mathbf{t}$. It follows directly that $\Gamma', D, \neg\Delta' \rightarrow P(u_1,\ldots,u_k)$ is provable in **LFO(ID,SLFP)**. By Lemma 4.5.10 (applied k times), we thus obtain that $\Gamma', D, \neg\Delta' \rightarrow P(t_1,\ldots,t_k)$ is provable in **LFO(ID,SLFP)**, as desired. Similarly, we have that if $P^{M_\omega}([t_1],\ldots,[t_k]) = \mathbf{f}$, then $\Gamma', D, \neg\Delta' \rightarrow \neg P(t_1,\ldots,t_k)$ is provable in **LFO(ID,SLFP)**.

Definition 4.5.18 *Define a two-valued interpretation N_ω for Σ by:*

- *N_ω is a two-valued Σ-interpretation;*
- *N_ω has the same domain as M_ω, namely, $Terms(\Sigma)/\sim$, the set of $\sim$-equivalence classes of Σ-terms;*
- *for any constant symbol c, $c^{N_\omega} = [c]$;*
- *for any function symbol f in Σ of arity k, $f^{N_\omega}([t_1],\ldots,[t_k]) = [f(t_1,\ldots,t_k)]$;*
- *for any open predicate symbol O in Σ of arity k, $O^{N_\omega}([t_1],\ldots,[t_k]) = \mathbf{t} \Leftrightarrow \exists u_1,\ldots,u_k.t_1 \sim u_1,\ldots,t_k \sim u_k$ and $O(u_1,\ldots,u_k) \in \Gamma_\omega$;*
- *for any definition D occurring in Γ_ω, defining a well-founded induction of D extending $N_\omega|_{Open(D)}$ as a sequence $(I^\xi)_{\xi\leq\alpha}$ of three-valued Σ-interpretation such that:*

- *for every defined predicate symbol* P, $P^{I^0} = \mathbf{u}$,
- *for each limit ordinal* $\lambda \leq \alpha$, $I^\lambda = lub_{\leq_p}(\{I^\xi \mid \xi < \lambda\})$, *and*
- *for every ordinal* ξ, $I^{\xi+1}$ *relates to* I^ξ *in one of the following ways:*
  - $I^{\xi+1} = I^\xi[P(t_1, \ldots, t_k)/\mathbf{t}]$ *for some* $P(t_1, \ldots, t_k)$ *such that* $P(t_1, \ldots, t_k)$ *is unknown in* I^ξ *and* $\varphi_P^{I^\xi}(t_1, \ldots, t_k) = \mathbf{t}$*;*
  - $I^{\xi+1} = I^\xi[U/\mathbf{f}]$ *, where* U *is the maximal unfounded set of* D *in* I^ξ*.*
- *for any defined predicate symbol* P *of* D *in* Σ *of arity* k*,*
  - $P^{N_\omega}([t_1], \ldots, [t_k]) = \mathbf{t} \Leftrightarrow \exists u_1, \ldots, u_k . t_1 \sim u_1, \ldots, t_k \sim u_k$ *and* $P^{I^\alpha}(u_1, \ldots, u_k) = \mathbf{t}$*;*
  - $P^{N_\omega}([t_1], \ldots, [t_k]) = \mathbf{f} \Leftrightarrow \exists u_1, \ldots, u_k . t_1 \sim u_1, \ldots, t_k \sim u_k$ *and* $P^{I^\alpha}(u_1, \ldots, u_k) = \mathbf{f}$*;*
- *for any other defined predicate symbol* P *in* Σ *of arity* k *that is not defined in any definition* D *occurring in* Γ_ω*,*
  - $P^{N_\omega}([t_1], \ldots, [t_k]) = \mathbf{t} \Leftrightarrow \exists u_1, \ldots, u_k . t_1 \sim u_1, \ldots, t_k \sim u_k$ *and* $P(u_1, \ldots, u_k) \in \Gamma_\omega$*;*
  - $P^{N_\omega}([t_1], \ldots, [t_k]) = \mathbf{f}$*, otherwise;*

Then N_ω *is said to be a two-valued extension of* M_ω*.*

Note that every definition occurring in the sequent $\Gamma_\omega \to \Delta_\omega$ is total. Hence, with the fact that $N_\omega|_{Open(D)}$ is two-valued, such a two-valued extension N_ω of M_ω exists. Notice that the properties of M_ω, as stated in Proposition 4.5.8, 4.5.9, hold for N_ω as well.

The following proposition is a direct consequence of the definition of N_ω.

Proposition 4.5.19 *For any predicate symbol* P *in* Σ *of arity* k, $P^{M_\omega}([t_1], \ldots, [t_k]) \leq_p P^{N_\omega}([t_1], \ldots, [t_k])$.

Lemma 4.5.20 *There exists a two-valued interpretation* W_ω *that extends* M_ω*, as defined in Definition 4.5.18, such that for every definition* D *occurring in* Γ_ω *and for every defined predicate* P *of* D*, if* $P(t_1, \ldots, t_k) \in \Gamma_\omega$*, then* $P^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ *and if* $P(t_1, \ldots, t_k) \in \Delta_\omega$*, then* $P^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{f}$*.*

Proof. Suppose that there does not exist such a W_ω. Then for every W_ω that extends M_ω, either for some defined atom $P(t_1, \ldots, t_k) \in \Gamma_\omega$, $P^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{f}$ or for some defined atom $P(t_1, \ldots, t_k) \in \Delta_\omega$, $P^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$.

- Assume that there exists a defined atom $P(t_1, \ldots, t_k)$ such that $P(t_1, \ldots, t_k) \in \Gamma_\omega$ and $P^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{f}$. For an arbitrary set of open literals Π such

that Π is the set of all open literals of D which do not occur in $\Gamma' \cup \Delta'$ and $W_\omega \models \bigwedge \Pi$, by the similar proof to Lemma 4.5.17, we can have that $\Gamma', \neg\Delta', \Pi, D \models \neg P(t_1, \ldots, t_k)$ is provable in **LFO(ID,SLFP)**. Moreover, by the negation rules and weakening rules, it is obtained that $\Gamma_\omega, \Pi \to \Delta_\omega$ is provable in **LFO(ID,SLFP)**.

- For the other case, assume that there exists a defined atom $P(t_1, \ldots, t_k)$ such that $P(t_1, \ldots, t_k) \in \Delta_\omega$ and $P^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$. By the similar proof to the above case, we can have that $\Gamma_\omega, \Pi \to \Delta_\omega$ is provable in **LFO(ID,SLFP)** as well.

Since Π is an arbitrary set of all open literals of D which do not occur in $\Gamma_\omega \cup \neg\Delta_\omega$, by using the cut rule multiple times, we have that $\Gamma_\omega \to \Delta_\omega$ is provable in **LFO(ID,SLFP)**, which is a contradiction. So there exists a desired two-valued interpretation W_ω that extends M_ω.

Lemma 4.5.21 *If $\varphi \in \Gamma_\omega$ then $W_\omega \models \varphi$, and if $\varphi \in \Delta_\omega$ then $W_\omega \not\models \varphi$.*

Proof. By induction on the structure of φ.

- **Case** $\varphi = P(t_1, \ldots, t_k)$. There are two subcases to consider: either P is an open predicate or a defined predicate.
  - **Subcase** $P = O$, where O is an open predicate symbol of Σ. Suppose $O(t_1, \ldots, t_k) \in \Gamma_\omega$. As $t_i \sim t_i$ for all $i \in \{1, \ldots, k\}$ by ($\sim$Refl), we thus have $O^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ by definition of W_ω, i.e., $W_\omega \models O(t_1, \ldots, t_k)$, as desired.

    Now suppose $O(t_1, \ldots, t_k) \in \Delta_\omega$ and suppose for contradiction that $W_\omega \models O(t_1, \ldots, t_k)$, i.e., $O^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ holds. Then there exist terms $u_1, \ldots, u_k$ such that $t_1 \sim u_1, \ldots, t_k \sim u_k$ and $O(u_1, \ldots, u_k) \in \Gamma_\omega$. So there is a proof of $\Gamma_\omega \to O(u_1, \ldots, u_k)$ (via an application of axiom). By Lemma 4.5.10 (applied k times), it follows that there is a proof of $\Gamma_\omega \to O(t_1, \ldots, t_k)$. Since $O(t_1, \ldots, t_k) \in \Delta_\omega$, there is a proof of $\Gamma_\omega \to \Delta_\omega$, which is a contradiction. Hence $W_\omega \not\models O(t_1, \ldots, t_k)$, as desired.
  - **Subcase** $P = P_i$, where P_i is a defined predicate in Σ. If D occurs in Γ_ω, $P_i \in Def(D)$ and $P_i(t_1, \ldots, t_k) \in \Gamma_\omega$ then by Lemma 4.5.20, ${P_i}^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$, i.e., $W_\omega \models P_i(t_1, \ldots, t_k)$, as desired. Otherwise, suppose that D does not occur in Γ_ω, $P_i \in Def(D)$ and $P_i(t_1, \ldots, t_k) \in \Gamma_\omega$. As $t_i \sim t_i$ for all $i \in \{1, \ldots, k\}$ by ($\sim$Refl), we thus have ${P_i}^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ by definition of W_ω, i.e., $W_\omega \models P_i(t_1, \ldots, t_k)$, as desired.

Now suppose that D occurs in Γ_ω, $P_i \in Def(D)$ and $P_i(t_1, \ldots, t_k) \in \Delta_\omega$. Then again by the Lemma 4.5.20, it is obtained that ${P_i}^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{f}$, i.e., $W_\omega \not\models P_i(t_1, \ldots, t_k)$, as desired. Otherwise, suppose that D does not occur in Γ_ω, $P_i \in Def(D)$ and $P_i(t_1, \ldots, t_k) \in \Delta_\omega$, and suppose for contradiction that $W_\omega \models P_i(t_1, \ldots, t_k)$, i.e., ${P_i}^{W_\omega}([t_1], \ldots, [t_k]) = \mathbf{t}$ holds. Then there exist terms $u_1, \ldots, u_k$ such that $t_1 \sim u_1, \ldots, t_k \sim u_k$ and $P_i(u_1, \ldots, u_k) \in \Gamma_\omega$. So there is a proof of $\Gamma_\omega \to P_i(u_1, \ldots, u_k)$ (via an application of axiom). By Lemma 4.5.10 (applied k times), it follows that there is a proof of $\Gamma_\omega \to P_i(t_1, \ldots, t_k)$. Since $P_i(t_1, \ldots, t_k) \in \Delta_\omega$, there is a proof of $\Gamma_\omega \to \Delta_\omega$, which is a contradiction. Hence $W_\omega \not\models P_i(t_1, \ldots, t_k)$, as desired.

- **Case** $\varphi = t_1 = t_2$. If $t_1 = t_2 \in \Gamma_\omega$ then we have $t_1 \sim t_2$ (by ($\sim$Base)) and thus $[t_1] = [t_2]$. By Proposition 4.5.8 it follows that ${t_1}^{W_\omega} = {t_2}^{W_\omega}$ and so $W_\omega \models t_1 = t_2$.

  Now suppose that $t_1 = t_2 \in \Delta_\omega$, and suppose for contradiction that $W_\omega \models t_1 = t_2$, i.e., ${t_1}^{W_\omega} = {t_2}^{W_\omega}$. By Proposition 4.5.8, $[t_1] = [t_2]$ and so $t_1 \sim t_2$. Now observe that $\Gamma_\omega \to t_1 = t_1$ is provable via an application of axiom. Hence by Lemma 4.5.10, $\Gamma_\omega \to t_1 = t_2$ is also provable. But since $t_1 = t_2 \in \Delta_\omega$, we would then have a proof of $\Gamma_\omega \to \Delta_\omega$, which is an contradiction. Hence $W_\omega \not\models t_1 = t_2$.

- **Case** $\varphi = \neg\varphi'$. If $\neg\varphi' \in \Gamma_\omega$, then by the construction of $\Gamma_\omega \to \Delta_\omega$ (c.f. Definition 4.5.5), there is an $i \geq 0$ such that for all $j \geq i$, $\neg\varphi' \in \Gamma_j$. Furthermore, it follows that $\varphi' \in \Delta_j$ for some $j \geq i$ and hence $\varphi' \in \Delta_\omega$. By induction hypothesis, we thus have $W_\omega \not\models \varphi'$, i.e., $W_\omega \models \neg\varphi$, as desired.

  Now suppose $\neg\varphi' \in \Delta_\omega$. By construction of $\Gamma_\omega \to \Delta_\omega$, $\varphi' \in \Gamma_\omega$ so by the induction hypothesis we have $W_\omega \models \varphi'$, i.e., $W_\omega \not\models \neg\varphi'$, as desired.

- **Case** $\varphi = \varphi_1 \wedge \varphi_2$. If $\varphi_1 \wedge \varphi_2 \in \Gamma_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, $\varphi_1 \in \Gamma_\omega$ and $\varphi_2 \in \Gamma_\omega$. By induction hypothesis $W_\omega \models \varphi_1$ and $W_\omega \models \varphi_2$, i.e., $W_\omega \models \varphi_1 \wedge \varphi_2$, as desired.

  Now if $\varphi_1 \wedge \varphi_2 \in \Delta_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$ we have $\varphi_1 \in \Delta_\omega$ or $\varphi_2 \in \Delta_\omega$. In the former case we have by the induction hypothesis $W_\omega \not\models \varphi_1$ whence it is clear that $W_\omega \not\models \varphi_1 \wedge \varphi_2$; the other case is similar.

- **Case** $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \supset \varphi_2$. These cases are similar to the previous case $\varphi = \varphi_1 \wedge \varphi_2$ above.

- **Case** $\varphi = \exists x\varphi'$. If $\exists x\varphi' \in \Gamma_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, we have $\varphi'[x/z] \in \Gamma_\omega$ for some variable z, whence $W_\omega \models \varphi'[x/z]$ by the induction hypothesis. It follows directly that $W_\omega \models \exists x\varphi'$.

  Now suppose $\exists x\varphi' \in \Delta_\omega$, and observe that by construction of $\Gamma_\omega \to \Delta_\omega$ there is an $i \geq 0$ such that $\exists x\varphi' \in \Delta_j$ for all $j \geq i$. Now consider an

arbitrary term t of Σ and note that the element $\langle \exists x\varphi', t\rangle$ appears infinitely often on the schedule $(E_i)_{i\geq 0}$ according to which $\Gamma_\omega \to \Delta_\omega$ is constructed. So there is a $j \geq i$ such that $E_j = \langle \exists x\varphi', t\rangle$ and thus we have $\varphi'[x/t] \in \Delta_\omega$. As t was chosen arbitrarily, it follows that for every term t, $\varphi'[x/t] \in \Delta_\omega$. So by the induction hypothesis $W_\omega \not\models \varphi'[x/t]$ for every term t. Suppose for contradiction that $W_\omega \models \exists x\varphi'$. Then for some $t \in Term(\Sigma)$, we would have $W_\omega \models \varphi'[x/t]$, which contradicts our induction hypotheses. Hence $W_\omega \not\models \exists x\varphi'$.

- **Case** $\varphi = \forall x\varphi'$. This case is similar to the case $\varphi = \exists x\varphi'$ above.

- **Case** $\varphi = [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi$.
If $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k) \in \Gamma_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k) \in \Gamma_\omega$ for some m. By the induction hypothesis, we have $W_\omega \models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k)$ for some m and hence, it holds that $W_\omega \models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k)$, as desired.

  Now if $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k) \in \Delta_\omega$ then by construction of $\Gamma_\omega \to \Delta_\omega$, $[LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k) \in \Delta_\omega$ for all m. By the induction hypothesis, we have $W_\omega \not\models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]_m\psi(t_1,\ldots,t_k)$ for all m and hence, it holds that $W_\omega \not\models [LFP_{X_1(\overline{x_1}),\ldots,X_n(\overline{x_n})}(\theta_1,\ldots,\theta_n)]\psi(t_1,\ldots,t_k)$, as desired.

- **Case** $\varphi = D$. Suppose that $D \in \Gamma_\omega$. By definition of W_ω, we thus have that W_ω is a well-founded model of D extending $W_\omega|_{Open(D)}$, as desired.

  Now suppose that $D \in \Delta_\omega$ and suppose for contradiction that $W_\omega \models D$. By construction of $\Gamma_\omega \to \Delta_\omega$, we have that $D^R \in \Gamma_\omega$ and $\forall\overline{x}(P(\overline{x}) \equiv P^R(\overline{x})) \in \Delta_\omega$ for some defined predicate P of D. By definition of W_ω, we thus have that $W_\omega \models D^R$. By induction hypothesis, we have that $W_\omega \not\models \forall\overline{x}(P(\overline{x}) \equiv P^R(\overline{x}))$. However, by the construction of D^R and the fact that $W_\omega \models D$ and $W_\omega \models D^R$, it is obvious that $W_\omega \models \forall\overline{x}(P(\overline{x}) \equiv P^R(\overline{x}))$ for each defined predicate P of D, which is the desired contradiction. Hence, it holds that $W_\omega \not\models D$, as desired.

Now we state the main completeness theorem for **LFO(ID,SLFP)**.

Theorem 4.5.22 (Completeness of LFO(ID,SLFP)) *Let Γ and Δ be arbitrary sets of SFO(ID,SLFP) formulas. If $\Gamma \to \Delta$ is valid, then $\Gamma \to \Delta$ is provable in* **LFO(ID,SLFP)**.

Proof. Suppose for contradiction that $\Gamma \to \Delta$ is valid, but not provable in **LFO(ID,SLFP)**, and let $\Gamma_\omega \to \Delta_\omega$ be the limit sequent for $\Gamma \to \Delta$ (c.f. Definition 4.5.5) with counter-interpretation W_ω (c.f. Definition 4.5.18,

Lemma 4.5.20). By Lemma 4.5.21, we have $W_\omega \models \varphi$ for all $\varphi \in \Gamma_\omega$ and $W_\omega \not\models \varphi$ for all $\varphi \in \Delta_\omega$. As $\Gamma \subseteq \Gamma_\omega$ and $\Delta \subseteq \Delta_\omega$, we thus have $W_\omega \models \varphi$ for all $\varphi \in \Gamma$ and $W_\omega \not\models \varphi$ for all $\varphi \in \Delta$.

In other words, we have constructed a two-valued Σ-interpretation W_ω such that $\Gamma \to \Delta$ is not true in W_ω and we have the required contradiction. Hence $\Gamma \to \Delta$ is provable in **LFO(ID,SLFP)**.

4.6 Conclusions and related work

In this chapter, we have presented a proof system for FO(ID) based on least fixpoint logic by introducing the logic FO(ID,SLFP) and extending Gentzen's sequent calculus for first order logic to the proof system **LFO(ID,SLFP)** for FO(ID,SLFP). The main technical results were the soundness theorem of **LFO(ID,SLFP)** for a slightly restricted fragment of FO(ID) and the completeness theorem of **LFO(ID,SLFP)** for a more restricted fragment of FO(ID), namely $\mathcal{SFO}(ID, SLFP)$.

Related work is provided by Brotherston in [26]. He introduced the language $\mathrm{FOL}_{\mathrm{ID}}$ of first order logic with the schema for inductive definitions, which is based upon Martin-Löf's "ordinary production" [119] and developed a proof system which is sound and complete with respect to a standard model in $\mathrm{FOL}_{\mathrm{ID}}$. A similar work to Brotherston's is studied by Hagiya and Sakurai in [83]. They proposed to interpret a (stratified) logic program as iterated inductive definitions of Martin-Löf and developed a proof theory which is sound with respect to the perfect model, and hence, the well-founded semantics of logic programming. Actually, both the $\mathrm{FOL}_{\mathrm{ID}}$ and the stratified logic programs as iterative definitions can be generalized in FO(ID). A formal proof system based on tableau methods for analyzing computation for Answer Set Programming (ASP) was given as well by Gebser and Schaub [70]. As shown in [114], ASP is closely related to FO(ID). Their approach furnishes declarative and fine-grained instruments for characterizing operations as well as strategies of ASP-solvers and provides a uniform proof-theoretic framework for analyzing and comparing different algorithms.

Chapter 5

Extending FO(.) with induction and coinduction

5.1 Introduction

Coinduction is the constructive principle complementary to (monotone) induction. A set can be constructed by coinduction if and only if its complement can be constructed by induction. Equivalently, a (monotone) induction corresponds to least fixpoint construction while a coinduction corresponds to a greatest fixpoint construction. Least and greatest fixpoint constructions can be nested in each other. Fixpoint logic is a well-known formalism for defining sets and relations by nested inductive and coinductive constructions.

The lack of expressive power of first-order logic to represent recursion (including recursion through negation) has motivated its extension with fixpoint constructs. For example, the *μ-calculus* [97, 170], a class of temporal logics with explicit fixpoints, provides a conceptually advantageous framework for specifying and reasoning about Real-Time Systems [61]. In the context of databases, query languages have been extended with fixpoint constructs to represent inductively definable concepts [19, 3]. Also, description logics have been extended with such fixpoint constructs [33]. These logics can be viewed as well-behaved fragments of first-order logic with fixpoints [139, 142]. The availability of explicit least and greatest fixpoint constructs in these logics allows inductive and coinductive concepts to be expressed in a natural way. Also in these logics, several forms of induction have been modelled: e.g., monotone induction, partial fixpoint induction, inflationary fixpoint induction.

In this chapter, we introduce a concept of *fixpoint definitions (FDs)*, which is a reformulation of fixpoint constructs by applying rule-based *syntactic sugar*. These fixpoint definitions use the format of LP-like rules which will enable us more easily to link it with logic programming notation and the notation for inductive definitions in FO(ID). We extend FO(.) with fixpoint definitions to define a new logic FO(FD). In the resulting logic, almost all kinds of inductions can be expressed as well. We investigate connections between the fixpoint definitions and non-monotone inductive definitions in FO(ID) by presenting equivalence preserving transformations of non-monotone inductive definitions to fixpoint definitions. It turns out that all kinds of inductive definitions in FO(ID) can be expressed in FO(FD). Meanwhile, due to the allowance of greatest fixpoints in FO(FD), coinduction can be represented in FO(FD). Thus, some concepts, e.g., infinite structures and co-recursion [16], which cannot be defined in FO(ID) through the well-founded way, can be handled naturally in FO(FD).

This chapter is structured as follows. Section 5.2 introduces induction and coinduction, which are the key concepts throughout the text. In Section 5.3, we trace the general history of fixpoint theory in Computer Science. Section 5.4 illustrates the relationship between various fixpoint logics and FO(ID). In Section 5.5, we introduce the basic concepts, which are used through the fixpoint theory. Section 5.6 introduces the concept of fixpoint definitions and formalizes the logic FO(FD). In Section 5.7, we investigate the relationship between FO(FD) and FO(ID) by providing translations from non-monotone inductive definitions in FO(ID) to fixpoint definitions and comparing FO(FD) with FO(ID) in terms of expressive power. Finally, we compare FO(FD) and coinductive logic programming in Section 5.8. We conclude in Section 5.9.

5.2 Induction and coinduction

Recently *coinduction* has been introduced as a technique for reasoning about unfounded sets [88], behavioral properties of programs [16, 77], and proving liveness properties in model checking [112]. Coinduction also serves as the foundation for lazy evaluation [79] and type inference [145] in functional programming as well as for interactive computing [78, 180].

Coinduction is the dual of induction. Induction corresponds to well-founded structures that start from a basis which serve as the foundation for building more complex structures. For example, natural numbers are defined inductively via the base element zero and the successor function. Inductive definitions have three components: initiality, iteration and minimality [78]. Thus, the inductive definition of list of numbers is as follows:

1. [ ] (empty list) is a list (initiality);

2. [H|T] is as a list if T is a list and H is some number (iteration); and,
3. nothing else is a list (minimality).

Minimality implies that infinite-length lists of numbers are not members of the inductively defined set of lists of numbers. Inductive definitions correspond to least fixpoint interpretations of recursive definitions.

Coinduction eliminates the initiality condition and replaces the minimality condition with maximality. Thus, the coinductive definition of a list of numbers is:

1. [H|T] is as a list if T is a list and H is some number (iteration); and,
2. the set of lists is the maximal set of such lists.

There is no base case in coinductive definitions, and while this may appear circular, the definition is well formed since coinduction corresponds to the greatest fixpoint interpretation of recursive definitions (recursive definitions for which greatest fixpoint interpretation is intended are termed corecursive definitions). Thus, the set of lists under coinduction is the set of all infinite lists of numbers (no finite lists are contained in this set). Note, however, that if we have a recursive definition with a base case, then under coinductive interpretation, the set defined will contain both finite and infinite-sized elements, since in this case the greatest fixpoint will also contain the least fixpoint. In the context of logic programming, in the presence of coinduction, proofs may be of infinite length. A coinductive proof essentially is an infinite-length proof.

However, the use of coinduction can lead to values which are "too infinite". For instance, a non-trivial binary relation defined as a coinductive inference system cannot include the rule of transitivity, because a coinductive reading of transitivity would imply that every element is related to every other (to see this, build an infinite derivation consisting solely of uses of transitivity). As pointed out in [67] this is unfortunate, because without transitivity conceptually unrelated rules may have to be merged or otherwise modified, in order to ensure that transitivity can be proved as a derived property. Gapeyev et al. [67] give the example of subtyping for records, where a dedicated rule of transitivity ensures that one can give separate rules for depth subtyping (which states that a record field type can be replaced by a subtype), width subtyping (which states that new fields can be added to a record), and permutation of record fields.

Fortunately this problem can be solved. The problem stems from a coinductive reading of transitivity, and it can be solved by reading the rule of transitivity inductively, and only using coinduction where it is necessary. This idea is illustrated by using mixed induction and coinduction to define a subtyping relation

for recursive types. The rule which defines when a function type is a subtype of another is defined coinductively (following [25] and [67]), while the rule of transitivity is defined inductively.

The technique of mixing induction and coinduction is not limited to defining transitive inference systems. It is generally useful when defining sets of tree-shaped values (like the typing derivations above) where some values are guaranteed to have finite depth (typing derivations consisting solely of the rules of transitivity and reflexivity, for instance), and other values are allowed to have infinite depth.

5.3 Fixpoints in computer science

This section is strongly based on the work in [153].

Induction and coinduction are intimately related to fixpoints. We therefore also examine fixpoints. We do not attempt, however, to trace the general history of fixpoint theory in Mathematics. Instead, we concentrate on Computer Science, and recall some papers that well witness the introduction of induction, coinduction and fixpoint theory for the design and analysis of programming languages. The Knaster-Tarski Theorem [174], which is stated in Section 5.5, about the existence of least and greatest fixpoints for a monotone function on a complete lattice is the starting point for all the work we mention.

The earliest uses of fixpoints in Computer Science, in the form of least fixpoints, can be found in: recursive function theory, see for instance Rogers's book [151] and references therein; formal language theory, as in the work of Arden [7] and Ginsburg and Rice [76]. However, distinguishing Computer Science from recursive function theory, the importance of fixpoints in Computer Science really comes up only at the end of 1960s, with four independent papers, roughly at the same time, by Dana Scott and Jaco de Bakker [162], Hans Bekič [17], David Park [139], and Antoni Mazurkiewicz [120].

The above four papers bring out the importance of least fixpoints for the semantics of programs, the relevance of lattice theory and the Knaster-Tarski Theorem, and propose various rules for reasoning about least fixpoints. Programs take the form of recursive function definitions or of flowcharts. Further, [162] paves the way for the fundamental work on denotational semantics by Scott and Strachey in Oxford in the 1970s, where least fixpoints, and continuity of functions, are essential. Influential on the above four papers are earlier works on program correctness and on uses of the "paradoxical combinator" Y of the λ-calculus, notably papers by Landin such as [99], by McCarthy such as [122, 123], and by Floyd such as [64]. For instance, McCarthy [122, 123] proposes the first method for proving properties of recursive programs, called *recursion induction*; variants and

stronger versions of the method are formulated in [162], [17], and [139]. Also the fixpoint properties of the Y combinator of the λ-calculus have been known for a long time, but the precise mathematical meaning of Y as fixpoint remains unclear until Scott works out his theory of reflexive domains, at the end of 1969 [156, 157]; see [140]. Another relevant paper is [158], in which fixpoints appear but which precedes the discovery of reflexive domains. We may also recall James H. Morris, who earlier [131] had proved a minimal fixpoint property for the Y combinator; in the same document, Morris had considered the relationship between least fixpoints and functions computed by first-order recursive definitions of programs.

During the 1970s, further fixpoint techniques and rules are put forward. A number of results on fixpoints and induction rules, and the basic theory of continuous functions, are due to Scott, e.g., [159, 160, 161]. On uses of least fixpoints in semantics and in techniques for program correctness, we should also mention the work of de Bakker and his colleagues, e.g., [12, 10, 11]; the Polish school, with Mazurkiewicz, Blikle, and colleagues, e.g., [120, 121, 21]; the work of Ugo Montanari and colleagues in Pisa, such as [75] that contains notions of observations and equivalence of representations in abstract data types that today we recognize as related to fixpoints via the concept of finality. Other references to the early works on least fixpoints can be found in Zohar Manna's textbook [113].

The above works all deal with least fixpoints. Greatest fixpoints, and related coinductive techniques, begin to appear as well in the 1970s. It is hard to tell what is the first appearance. One reason for this is that the rules for greatest fixpoints are not surprising, being the dual of rules for least fixpoints that had already been studied. The first to make explicit and non-trivial use of greatest fixpoints is David Park, who, throughout the 1970s, works intensively on fairness issues for programs. He was using rather sophisticated constructs, possibly involving alternation of least and greatest fixpoints. Park discusses his findings in several public presentations. A late overview paper is [143].

Other early interesting uses of greatest fixpoints are made by the following authors. Mazurkiewicz [121] studies properties of computations from processes, where processes are modeled via forms of LTSs; the properties studied include divergence and termination. The way in which Mazurkiewicz defines divergent states (i.e., the states from which a computation may not terminate) and the technique proposed to prove divergence of states are coinductive, though—as in his earlier paper [120]—there is no explicit reference to fixpoint theory.

Edmund Clarke [35] shows that the correctness proofs for Floyd-Hoare axiom systems—deductive systems for partial correctness based on invariance assertions intensively investigated in the 1970s—could be elegantly formalised by means of fixpoint theory, whereby: program invariants become greatest fixpoints; completeness of a system becomes the proof of the existence of a fixpoint for an appropriate functional; and soundness is derived from the maximality of such

fixpoint. Thus soundness is a coinductive proof. Willem-Paul de Roever [46] strongly advocates the coinduction principles as a proof technique. De Roever uses the technique to reason about divergence, bringing up the duality between this technique and inductive techniques that had been proposed previously to reason on programs.

Greatest fixpoints and coinduction are implicit in a number of earlier works in the 1960s and 1970s. Important examples, with a huge literature, are the works on unification, for instance on structural equivalence of graphs, and the works on invariance properties of programs. Fixpoints are also central in *stream processing* systems (including *data flow systems*). The introduction of streams in Computer Science is usually attributed to Peter Landin, in the early 1960s (see [100, 101]). However, fixpoints are explicitly used to describe stream computations only after Scott's theory of domain, with the work of Gilles Kahn [93].

The first appearance of the term "coinduction" is in Barwise and Etchemendy's 1987 book [15]. The term "coinduction" also occurs in earlier papers and textbooks in Mathematics (e.g., [132]), but simply to indicate the complement of an inductively defined structure. In Computer Science, the term is widely used after Milner and Tofte [128], who use coinduction to prove the soundness of a type system, and describe coinduction to explain the analogy between the technique for types in their paper and the bisimulation techniques. The main objective of the paper is indeed to advocate the proof technique and to suggest the name coinduction for it. Given the duality with induction, the term "coinduction" is so natural that it has become part of computer science folklore and that it is hard to trace back its origin.

5.4 Fixpoint logics and FO(ID)

It is well-known that, in general, inductive definitions cannot be expressed in first-order logic (FO). Yet, inductively defined concepts are often very useful in practice. For example, in the context of databases, query languages have been extended with fixpoint constructs to represent inductively definable concepts. Also, description logics have been extended with such fixpoint constructs. The lack of expressive power of first-order logic to represent recursion (including recursion through negation) has motivated its extension with fixpoint constructs. In these logics, several forms of induction have been modelled: e.g., monotone induction, partial fixpoint induction, inflationary fixpoint induction. However, there are forms of non-monotone induction which are quite common in mathematics that are not modelled well by fixpoint logics. In particular, inductions on well-founded orders and iterated inductions [98, 29] are not handled naturally in any of the fixpoint logics.

FO(ID) is a coherent and conceptually clean integration of classical logic and logic programming. These two formalisms are usually viewed as serving such thoroughly different representational and computational purposes, that an integration seems impossible or even undesirable. But if we see logic programs as declarative representations of inductive definitions, then it is natural to integrate logic programs with classical logic, in order to compensate for the latter's representational weakness on inductive definability. This enterprise is similar to the extension FO(LFP) of classical logic with monotone fixpoints, or to the extension FO(IFP) with inflationary fixpoints. The main difference with fixpoint logics is that it incorporates other forms of non-monotone induction, which arise much more often in mathematics and common-sense reasoning.

The well-founded model construction was first presented in [72] but the link with iterated induction and induction over well-founded orders was laid only later in [49, 51]. Nevertheless, the view of logic programs as representations of definitions has been implicit in many studies on the semantics of logic programming, for example in the completion semantics [34], in the *standard* semantics of stratified logic programs [5], in the original work of well-founded semantics [72], and in [155]. In [71], the well-founded semantics was extended to rule sets with arbitrary FO-bodies. Van Gelder presents this logic in the spirit of fixpoint logics, as a logic of alternating fixpoints and calls it *AFP*, for *Alternating Fixpoint Partial model*. This logic can be viewed as an ancestor of FO(ID).

Mathematical induction refers to a class of construction techniques used in mathematics. There, a set, or a set of sets, or a structure, is frequently defined as the result of an iterative constructive process, consisting of repeated applications of a basic set of operations. Mathematicians often describe such a construction by an *inductive definition*. The scientific study of the concept of inductive definition, as used in mathematical texts, is an important topic of mathematical logic. In this section, we discuss several *informal* forms of inductive definitions and how they have been formalized in fixpoint logics and FO(ID).

Mathematicians often follow certain linguistic conventions to phrase such definitions. For example, consider the following definition:

Definition 5.4.1 *The truth relation $\models$ between interpretations I and propositional sentences φ is defined by induction on the subformula order (or on the length) of formulas:*

- $I \models p$ *if* $p \in I$,
- $I \models \varphi \wedge \psi$ *if* $I \models \varphi$ *and* $I \models \psi$,
- $I \models \varphi \vee \psi$ *if* $I \models \varphi$ *or* $I \models \psi$,
- $I \models \neg\varphi$ *if* $I \not\models \varphi$.

This definition starts with the phrase that the defined set "*is defined by induction (on some order)*" followed by a set of basic and inductive rules, where the latter type of rules specify when to add elements to the defined set, given the presence or absence of other elements.

In most inductive definitions in mathematics, the inductive rules are all *monotone*, meaning that they add elements to the defined sets given the presence of other elements in the set. For example:

Definition 5.4.2 *The transitive closure T_G of a directed graph G is inductively defined by the following rules:*

- $(x, y) \in T_G$ *if* $(x, y) \in G$;
- $(x, y) \in T_G$ *if for some vertex* z, $(x, z) \in T_G$ *and* $(z, y) \in T_G$.

Other typical examples of monotone inductive definitions are the definition of a subgroup generated by a set of group elements, or the definitions of a term, formula, etc. in logic. A fundamental application in logic is that of the deductive closure $Cn(T)$ of a propositional or first-order logic theory T which is inductively defined by the formulas of T and a sound and complete set of inference rules.

The set defined by such definitions can be characterized in a *downward*, nonconstructive way, as the least set closed under application of the rules, i.e., as the intersection of all sets for which the conclusion of each rule is satisfied whenever the condition is satisfied. Alternatively, the defined set can also be characterized in an *upward*, constructive way as the limit of a process starting from the empty set and iteratively applying rules until saturation occurs. Although these characterizations are very different, it is a fundamental fact of monotone induction that both views correspond.

Monotone induction has been extensively studied from an expressivity-theoretic point of view and is a key concept in recursion theory. Its study was started by [147] and was continued in many later studies such as [168, 132, 2]. In these formal studies, a monotone inductive definition was often formalized by a formula $\varphi(\bar{x}, X)$ where X is a predicate variable of arity n with only positive occurrences in φ and $\bar{x}$ a tuple of n variables. This formula encodes the set of rules of an inductive definition by specifying all the conditions under which tuple $\bar{x}$ belongs to the defined predicate X. For instance, for the transitive closure example above we have:

$$\varphi_{trans}((x, y), T_G) := G(x, y) \vee \exists z(T_G(x, z) \wedge T_G(z, y)).$$

Given a structure I, the formula $\varphi(\bar{x}, X)$ characterises an operator $\Gamma_{\varphi(\bar{x},X)}$ mapping a relation R to the relation R' consisting of tuples $\bar{a}$ such that $\varphi(\bar{a}, R)$ is true in I. For positive formulas, this operator is monotone, i.e., preserves the

subset relation $\subseteq$, and has a least fixpoint, which is both the least set closed under application of the operator (i.e., the least pre-fixpoint of $\Gamma_{\varphi(\bar{x},X)}$, i.e., the least R such that $\Gamma_{\varphi(\bar{x},X)}(R) \subseteq R$ and the fixpoint obtained by iterating the operator on the empty relation. A logic to represent monotone inductive definitions is the least fixpoint logic FO(LFP) (see, e.g., [59]).

Another form of non-monotone induction: Inflationary Induction.

In order to extend the theory of monotone inductive definitions to the class of all formulas, Moschovakis [133] proposed to associate with an arbitrary formula $\varphi(\bar{x}, X)$ (possibly non-positive) the operator $\Gamma'_{\varphi(\bar{x},X)}$, where

$$\Gamma'_{\varphi(\bar{x},X)}(R) := \Gamma_{\varphi(\bar{x},X)}(R) \cup R.$$

Operator $\Gamma'_{\varphi(\bar{x},X)}$ is not monotone, but it is *inflationary*, that is, for every R, $R \subseteq \Gamma'_{\varphi(\bar{x},X)}(R)$. Thus, by iterating this operator starting at the empty relation, an ascending sequence can be constructed. This sequence eventually reaches a fixpoint of $\Gamma'_{\varphi(\bar{x},X)}$. This fixpoint was later called the *inflationary fixpoint*, and the corresponding logic FO(IFP) was introduced [82]. This logic introduces inflationary, and its dual, deflationary, fixpoint constructs. The inflationary fixpoint logic played an important role in descriptive complexity theory and has been used to characterize the complexity class PTIME [87, 108, 176]. FO(IFP) is well-known to be a very expressive logic. Some applications of inflationary and deflationary induction were presented in [80].

Inflationary induction and iterated induction (as formalized in FO(ID)) are different extensions of monotone induction. Our argument to extend FO with induction over well-founded order and iterated induction rather than inflationary induction stems from knowledge representation. Induction over well-founded order and iterated induction have many applications in mathematics. It is useful to demonstrate that these forms of inductive definitions have also many applications outside mathematics, in knowledge representation and computational logic. In comparison, natural applications of inflationary induction in mathematics and computational logic seem to be quite rare. Although inflationary induction is expressive, in practice rephrasing even simple iterated inductive definitions through inflationary induction may be extremely difficult. The author of [71] discusses the case of the complement NT_G of the transitive closure of a graph G. The obvious way to define this in FO(ID) is:

$$\left\{ \begin{array}{c} \forall x \forall y (T_G(x,y) \leftarrow G(x,y) \vee \exists z (T_G(x,z) \wedge T_G(z,y))), \\ \forall x \forall y (NT_G(x,y) \leftarrow \neg T_G(x,y)) \end{array} \right\}.$$

This is a simple iterated inductive definition with two levels. On the other hand, according to [71], it was "a significant research achievement" when a set

of (function-free) logic programming rules (rules with a conjunction of literals as body) was discovered expressing the same relation under the inflationary semantics. The simplest known solution is a complex set of rules including several intermediate relations and it is far from obvious how computing the inflationary fixpoint of this program constructs the relation.

5.5 Formal Preliminaries

In this section, we introduce the basic concepts used through the fixpoint theory.

5.5.1 Orders, lattices, operators and fixpoints

A *pre-ordered set* is a structured set $\langle W, \leq \rangle$, where W is an arbitrary set and $\leq$ is a pre-order on W, i.e., a reflexive and transitive binary relation. As usual, $x < y$ is a shorthand for $x \leq y \wedge y \not\leq x$. A *pre-well-founded set* is a pre-ordered set where $\leq$ is a pre-order such that every non-empty set $S \subseteq W$ contains a *minimal element*, i.e., an element x such that for each $y \in S$, if $y \leq x$ then $x \leq y$. Equivalently, it is a set without infinite descending sequence of elements $x_0 > x_1 > x_2 > \ldots$.

A *partially ordered set*, or simply *poset*, is an asymmetric pre-ordered set $\langle W, \leq \rangle$, i.e., one such that $x \leq y$ and $y \leq x$ implies $x = y$. A *well-founded set* is a pre-well-founded poset.

A lattice is a poset $\langle L, \leq \rangle$ such that every finite set $S \subseteq L$ has a least upper bound $lub(S)$, the *supremum* of S, and a greatest lower bound $glb(S)$, the *infimum* of S. A lattice $\langle L, \leq \rangle$ is *complete* if *every* (not necessarily finite) subset of L has both a supremum and an infimum. Consequently, a complete lattice has a least element ($\bot$) and a greatest element ($\top$). An example of a complete lattice is the power set lattice $\langle \mathrm{Pow}(A), \subseteq \rangle$ of some set A. For any set S of elements of this lattice (i.e., for any set S of subsets of A), its least upper bound is the union of these elements, $lub(S) = \bigcup S$. Thus, the greatest element $\top$ of $\langle \mathrm{Pow}(A), \subseteq \rangle$ is $\bigcup \mathrm{Pow}(A)$, which is A. Similarly, $glb(S) = \bigcap S$, and the least element $\bot$ of this lattice is $\bigcap \mathrm{Pow}(A)$, which is $\emptyset$.

Given a lattice $\langle L, \leq \rangle$, an operator $\Gamma : L \to L$ is *monotone* with respect to $\leq$ if $x \leq y$ implies $\Gamma(x) \leq \Gamma(y)$. Operator Γ is non-monotone, if it is not monotone. A *pre-fixpoint* of Γ is a lattice element x such that $\Gamma(x) \leq x$. A *fixpoint* of Γ is a lattice element x such that $\Gamma(x) = x$. And a *post-fixpoint* of Γ is a lattice element x such that $\Gamma(x) \geq x$. The following theorem was obtained by Tarski in 1939 and is sometimes referred to as the Knaster-Tarski Theorem because it improves their earlier joint result. The theorem was published in [174], and it is one of the basic tools to study fixpoints of operators on lattices.

Theorem 5.5.1 (Existence of fixpoints) *[174] Every monotone operator over a complete lattice $\langle L, \leq \rangle$ has a complete lattice of fixpoints (and hence a least fixpoint $lfp(\Gamma)$ and greatest fixpoint $gfp(\Gamma)$).*

This least fixpoint $lfp(\Gamma)$ is the least pre-fixpoint of Γ and is the supremum of the sequence $(x^\xi)_\xi$ which is defined inductively

$$x^\xi := \Gamma(x^{<\xi}), \text{ and } x^{<\xi} := lub\{x^\eta \mid 0 \leq \eta < \xi\}.$$

Notice that $x^{<0}$ is, by definition, $\bot$.

This greatest fixpoint $gfp(\Gamma)$ is the greatest post-fixpoint of Γ and is the infimum of the sequence $(x^\xi)_\xi$ which is defined inductively

$$x^\xi := \Gamma(x^{<\xi}), \text{ and } x^{<\xi} := glb\{x^\eta \mid 0 \leq \eta < \xi\}.$$

Notice that $x^{<0}$ is, by definition, $\top$.

An operator Γ is *anti-monotone* if $x \leq y$ implies $\Gamma(y) \leq \Gamma(x)$. The square $\Gamma^2 = \Gamma \circ \Gamma$ of an anti-monotone operator is monotone.

An *oscillating pair* of an operator Γ is a pair (x, y) such that $\Gamma(x) = y$ and $\Gamma(y) = x$. An anti-monotone operator Γ in a complete lattice has a maximal oscillating pair (x, y), i.e., for any oscillating pair (x', y'), it holds that $x \leq x'$ and $y' \leq y$. Since (y, x) is also an oscillating pair, it follows that $x \leq y$. Moreover, since each fixpoint z of Γ corresponds to an oscillating pair (z, z), it follows that $x \leq z \leq y$. If in addition $x = y$ then x is the unique fixpoint of Γ.

The maximal oscillating pair (x, y) of Γ can be constructed by an alternating fixpoint computation. Define four sequences $(x^\xi)_\xi, (x^{<\xi})_\xi, (y^\xi)_\xi, (y^{<\xi})_\xi$ by the following transfinite induction:

- $x^{<\xi} = lub(\{x^\eta : \eta < \xi\})$,
- $x^\xi = \Gamma(y^{<\xi})$,
- $y^{<\xi} = glb(\{y^\eta : \eta < \xi\})$,
- $y^\xi = \Gamma(x^{<\xi})$.

Note that $x^{<0} = \bot$ and $y^{<0} = \top$. It can be shown that for each ξ, $x^{<\xi} \leq x^\xi \leq y^\xi \leq y^{<\xi}$. The following theorem holds.

Theorem 5.5.2 *[71] The sequence $(x^\xi)_\xi$ is ascending and its supremum is $lfp(\Gamma^2)$. The sequence $(y^\xi)_\xi$ is descending and its infimum is $gfp(\Gamma^2)$. The pair $(lfp(\Gamma^2), gfp(\Gamma^2))$ is the maximal oscillating pair of Γ.*

In the sequel, we denote the maximal oscillating pair of an anti-monotone operator G by $\mathsf{OSC}(G)$.

5.5.2 Lattice congruences

Let $\langle L, \leq \rangle$ be a complete lattice and let $\cong$ be an equivalence relation (i.e., a reflexive, symmetric and transitive relation) on L. For any $x \in L$, we denote its equivalence class $\{y \in L \mid x \cong y\}$ by $|x|$. The collection of equivalence classes is denoted by $|L|$. The relation $\cong$ can be extended to tuples: $(x_1, \ldots, x_n) \cong (y_1, \ldots, y_n)$ if $x_1 \cong y_1$ and ... and $x_n \cong y_n$. It is extended to subsets of L by defining for all S, $S' \subseteq L$: $S \cong S'$ if for each $x \in S$ there exists $x' \in S'$ such that $x \cong x'$ and vice versa, for each $x' \in S$ there exists $x \in S$ such that $x \cong x'$. We sometimes call an element of $\tilde{x} \in |L|$ a *witness* of $\tilde{x}$.

An equivalence relation $\cong$ on L is called a *lattice congruence* of $\langle L, \leq \rangle$ if for each pair $S, S' \subseteq L$, $S \cong S'$ implies that $lub(S) \cong lub(S')$ and $glb(S) \cong glb(S')$. We can define a binary relation $\leq$ on $|L|$: for all $\tilde{x}, \tilde{y} \in |L|$, define $\tilde{x} \leq \tilde{y}$ if for some $x \in \tilde{x}, y \in \tilde{y} : x \leq y$. It can be shown easily that if $\cong$ is a lattice congruence, then the structure $\langle |L|, \leq \rangle$ is a complete lattice.

A lattice congruence $\cong$ on L defines a pre-order $\leq_{\cong}$ on L, defined as $x \leq_{\cong} y$ iff $|x| \leq |y|$ iff $\exists x', y' : x \cong x' \leq y' \cong y$.

For any operator $\Gamma : L^m \to L^n$, we say that Γ preserves $\cong$ if for any pair of $\bar{x}, \bar{y} \in L^m, \bar{x} \cong \bar{y}$ implies $\Gamma(\bar{x}) \cong \Gamma(\bar{y})$. When an operator $\Gamma : L \to L$ preserves $\cong$, we define its quotient $|\Gamma|$ as the operator on $|L|$ mapping every $\tilde{x} \in |L|$ to $|\Gamma(x)|$ where x is an arbitrary witness of $\tilde{x}$. This definition can be extended to operators $\Gamma : L^m \to L^n$.

The following straightforward proposition describes the relationships between Γ and $|\Gamma|$.

Proposition 5.5.3 *Assume that $\Gamma : L \to L$ is an operator which preserves $\cong$.*

1. *If Γ is (anti-)monotone, then $|\Gamma|$ is (anti-)monotone.*
2. *If Γ is monotone then $|lfp(\Gamma)| = lfp(|\Gamma|)$ and $|gfp(\Gamma)| = gfp(|\Gamma|)$.*
3. *If Γ is anti-monotone, then $|\mathsf{OSC}(\Gamma)| = \mathsf{OSC}(|\Gamma|)$.*

This proposition has a straightforward extension to operators of more arguments.

5.5.3 Structure lattices

Structure lattices are sets of structures that extend a given structure. For a given vocabulary Σ and structure K_o such that $\Sigma_{K_o} \subseteq \Sigma$, define $\mathcal{S}^{\Sigma}_{K_o}$ as the set of Σ-structures that extend K_o, i.e., the set of Σ-structures I such that $I\,|_{\Sigma_{K_o}} = K_o$.

For any pair I_1, I_2 of Σ-structures, define $I_1 \sqsubseteq I_2$ if for each interpreted relation symbol X, $X^{I_1} \subseteq X^{I_2}$. This relation is reflexive and transitive. It is not asymmetric, since I_1 and I_2 may be identical on all relation symbols but different on constant or function symbols. However, if K_o interprets all function symbols of Σ, then $\sqsubseteq$ is a complete lattice order in the set $\mathcal{S}^{\Sigma}_{K_o}$. Its least element is the structure $\bot_{K_o} := K_o[\bar{X} : \emptyset]$ assigning the empty relations to all symbols X in $\Sigma \setminus \Sigma_{K_o}$ and its largest element $\top_{K_o}$ is the structure assigning the Cartesian product A^n to each n-ary symbol $X \in \Sigma \setminus \Sigma_{K_o}$. For any subset $S \in \mathcal{S}^{\Sigma}_{K_o}$, we denote its least upper-bound by $\sqcup S$ and its greatest lower-bound by $\sqcap S$.

The lattice $\langle \mathcal{S}^{\Sigma}_{K_o}, \leq \rangle$ contains many sub-lattices. In particular, for any structure K extending K_o such that $\Sigma_{K_o} \subseteq \Sigma_K \subseteq \Sigma$, $\langle \mathcal{S}^{\Sigma}_{K}, \sqsubseteq \rangle$ is a sub-lattice of $\langle \mathcal{S}^{\Sigma}_{K_o}, \sqsubseteq \rangle$.

5.6 A logic of fixpoint definitions

In this section, we extend classical first-order logic (FO) with fixpoint definitions (FDs) to formalize a new logic FO(FD), which can be viewed as an alternative extension of first-order logic with mixed induction and coinduction. Our motivation for this is that, first, from a practical point of view, fixpoint definitions extend the expressivity of FO(.) and allow to express a number of useful and practical concepts that cannot (or only very unpleasantly) be expressed using FO(ID); second, from a theoretical point of view, we want to study the relationship between the non-monotone induction in FO(ID) and the constructive principle underlying fixpoint definitions. The fixpoint definition construct in our new logic FO(FD) is a syntactic variant of standard alternating fixpoint logic and differs from it in two (minor) ways: an LP-like rule notation is used for expressing inductive and coinductive definitions, and the definitions define predicate constants rather than predicate variables. This modification facilitates the comparison and integration with FO(ID) and often improves knowledge representation.

5.6.1 Syntax

A *rule* over vocabulary Σ is an expression of the form:

$$\forall \bar{x}(P(\bar{x}) \leftarrow \Psi[\bar{x}]),$$

where P is a predicate symbol of Σ and $\Psi[\bar{x}]$ is an arbitrary first-order formula over Σ. P is known as the *head* of the rule and $\Psi[\bar{x}]$ is known as the *body* of the rule. The *defined predicate* of the rule is P. The connective $\leftarrow$ is called *definitional implication* and is to be distinguished from material implication $\supset$. We say that a predicate symbol occurs positively (negatively) in a formula if it occurs in the

scope of an even (odd) numbers of negations. A rule is *positive* in a set σ of predicate symbols if these symbols occur only positively in Ψ.

For a set $\mathcal{R}$ of rules, we denote $Def(\mathcal{R})$ as the set of defined predicates of its rules, and we denote $Open(\mathcal{R})$ as the set of all other symbols occurring in $\mathcal{R}$.

An arbitrary set $\mathcal{R}$ of rules can be transformed into an equivalent set of rules such that each defined predicate $P \in Def(\mathcal{R})$ is defined by exactly one rule, denoted by $\forall \bar{x}(P(\bar{x}) \leftarrow \varphi_P[\bar{x}])$. We refer to this rule as *the rule defining* P. $\varphi_P[\bar{x}]$ will be used to denote the body of the rule defining P. The transformation is obtained by merging all rules with the same head predicate into one rule as follows: $\{\forall \bar{x}(P(\bar{x}) \leftarrow \varphi_1[\bar{x}]), \ldots, \forall \bar{x}(P(\bar{x}) \leftarrow \varphi_n[\bar{x}])\}$ becomes $\forall \bar{x}(P(\bar{x}) \leftarrow \varphi_1[\bar{x}] \vee \ldots \vee \varphi_n[\bar{x}])$, for all defined predicates P, and thus $\varphi_P[\bar{x}] = \varphi_1[\bar{x}] \vee \ldots \vee \varphi_n[\bar{x}]$. Without loss of generality, in what follows we assume that there is only one rule $\forall \bar{x}(P(\bar{x}) \leftarrow \varphi_P[\bar{x}])$ in a set $\mathcal{R}$ of rules for every $P \in Def(\mathcal{R})$.

Definition 5.6.1 *We define a* least fixpoint definition *(LFD), respectively* greatest fixpoint definition *(GFD), over vocabulary Σ by simultaneous induction, as a finite expression $\mathcal{D}$ of the form*

$$\lfloor\ \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n\ \rfloor, \textit{ respectively}$$

$$\lceil\ \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n\ \rceil$$

with $0 \leq n, m$ such that:

1. *$\mathcal{R}$ is a set of rules over Σ.*
2. *Each Δ_i is a least fixpoint definition and each ∇_j is a greatest fixpoint definition, and they are called* subdefinitions *of $\mathcal{D}$.*

To express the remaining conditions, we need some auxiliary concepts and notations. For such an expression $\mathcal{D}$, we say that a predicate P is locally defined *in $\mathcal{D}$ if $P \in Def(\mathcal{R})$, and that P is* defined *in $\mathcal{D}$ if P is locally defined in $\mathcal{D}$ or defined in any of its subdefinitions $\Delta_1, \ldots, \nabla_n$. The set of defined predicates of $\mathcal{D}$ is denoted $Def(\mathcal{D})$. A symbol is* open *in $\mathcal{D}$ if it occurs in $\mathcal{D}$ and is not defined in it. The set of open symbols of $\mathcal{D}$ is denoted $Open(\mathcal{D})$.*

3. *Every defined symbol of $\mathcal{D}$ has only positive occurrences in $\mathcal{D}$, i.e., it has only positive occurrences in the bodies of the rules of $\mathcal{R}$ and in the subdefinitions Δ_i and ∇_j.*
4. *Each symbol $P \in Def(\mathcal{D})$ has exactly one local definition in $\mathcal{D}$. Formally, $\{Def(\mathcal{R}), Def(\Delta_1), \ldots, Def(\nabla_n)\}$ is a partition of $Def(\mathcal{D})$.*

5. *For every subdefinition $\mathcal{D}'$ of $\mathcal{D}$, $Open(\mathcal{D}') \subseteq Open(\mathcal{D}) \cup Def(\mathcal{R})$. In particular, a symbol defined in another subdefinition $\mathcal{D}'' \neq \mathcal{D}'$, does not occur in $\mathcal{D}'$.*

A fixpoint definition *is either a least fixpoint definition or a greatest fixpoint definition.*

Both LFD and GFD expressions are defined as trees with nodes containing rule sets and two types of children, namely LFD's and GFD's. Each defined predicate of the expression is locally defined in exactly one node of this graph. Moreover, a predicate locally defined in one node, has only positive occurrences in rule bodies in the node, its ancestors and its descendants, and does not occur at all in a sibling of the node or the siblings descendants.

Example 5.6.2 *We represent a transition graph on a set of vertices representing states by a binary predicate T. Let R represent a property of states, i.e., it is a unary predicate on vertices. We define P to represent the set of states that can reach a state satisfying R by:*

$$\left\lfloor \begin{array}{l} \forall x(P(x) \leftarrow R(x)) \\ \forall x(P(x) \leftarrow \exists y(T(x,y) \wedge P(y))) \end{array} \right\rfloor .$$

Example 5.6.3 *We represent a transition graph on a set of vertices representing states by a binary predicate T. Let R represent a property of states, i.e., it is a unary predicate on vertices. We define P to represent the set of states that are on an infinite path consisting of states satisfying R by:*

$$\left\lceil \ \forall x(P(x) \leftarrow R(x) \wedge \exists y(T(x,y) \wedge P(y))) \ \right\rceil .$$

Initially, all states are in the set defined by predicate P. After one iteration, all states in the set satisfy R and that, after one transition they reach a state in the set. Hence after i'th iterations, one has nodes for which there is a path consisting of i successive states satisfying R. In the limit, it represents a set of states that are on an infinite path consisting of R-states.

Example 5.6.4 *We represent a transition graph on a set of vertices representing states by a binary predicate T. Let R represent a property of states, i.e., it is a unary predicate on vertices. We want P to represent the set of states which are on a path that ends in an infinite sequence of states with property R. This set is defined by:*

$$\left\lceil \begin{array}{l} \forall x \ (P(x) \leftarrow Q(x)) \\ \left\lfloor \begin{array}{l} \forall x \ (Q(x) \leftarrow R(x) \wedge \exists y(T(x,y) \wedge P(y))) \\ \forall x \ (Q(x) \leftarrow \exists y(T(x,y) \wedge Q(y))) \end{array} \right\rfloor \end{array} \right\rceil .$$

An *FO(FD) formula* is either an FO formula or a fixpoint definition. An *FO(FD) theory* is a set of fixpoint definitions and FO sentences.

5.6.2 Semantics

Given two disjunct first-order vocabularies Σ and Σ', a Σ-interpretation I and a Σ'-interpretation I', the $\Sigma \cup \Sigma'$-interpretation mapping each element e of Σ to e^I and each $e \in \Sigma'$ to $e^{I'}$ is denoted by $I + I'$. When $\Sigma' \subseteq \Sigma$, we denote the restriction of a Σ-interpretation I to the symbols of Σ' by $I|_{\Sigma'}$. For a Σ-interpretation I, a truth value v and a domain atom $P(\overline{a})$, we denote by $I[P(\overline{a})/v]$ the Σ-interpretation that assigns v to $P(\overline{a})$ and corresponds to I for all other domian atoms. We extend this notation to sets of domain atoms, i.e., for a Σ-interpretation I, a truth value v and a set of domain atoms U, we denote by $I[U/v]$ the Σ-interpretation that assigns v to every $P(\overline{a}) \in U$ and corresponds to I for all other domain atoms.

With a set $\mathcal{R}$ of rules over Σ and a (partial) two-valued Σ-interpretation I interpreting at least all open symbols and no defined symbols, i.e., $\Sigma \cap Def(\mathcal{R}) = \emptyset$ and $Open(\mathcal{R}) \subseteq \Sigma$, there is a standard way of associating an operator $\Gamma_I^{\mathcal{R}}$ on the set of $Def(\mathcal{R})$-interpretations with the domain of I. For two such interpretations J, K, we define $\Gamma_I^{\mathcal{R}}(J) = K$ if for every $P \in Def(\mathcal{R})$, $K(P) = \{\bar{d} | I + J \models \varphi_P[\bar{d}]\}$.

If each defined symbol in $Def(\mathcal{R})$ has only positive occurrences in the body of a rule in $\mathcal{R}$, the operator $\Gamma_I^{\mathcal{R}}$ is monotone with respect to the standard truth order on interpretations and hence, it has least and greatest fixpoints in this set denoted $lfp(\Gamma_I^{\mathcal{R}})$, respectively $gfp(\Gamma_I^{\mathcal{R}})$. Importantly, if $I(P) \leq I'(P)$ for every symbol $P \in Open(\mathcal{R})$ with only positive occurrences in rule bodies of $\mathcal{R}$, then $lfp(\Gamma_I^{\mathcal{R}}) \leq lfp(\Gamma_{I'}^{\mathcal{R}})$ and $gfp(\Gamma_I^{\mathcal{R}}) \leq gfp(\Gamma_{I'}^{\mathcal{R}})$.

The semantics of the FO(FD) is an integration of standard FO semantics with fixpoint semantics of definitions.

We firstly define the semantics of fixpoint definitions. Given an expression $\mathcal{D}$ which might be an LFD or a GFD, and an $Open(\mathcal{D})$-interpretation I interpreting at least all open symbols of $\mathcal{D}$ and no defined ones. We define an operator $\Gamma_I^{\mathcal{D}}$ on the set of $Def(\mathcal{D})$-interpretations with domain $dom(I)$. This operator is monotone with respect to the standard truth order on interpretations and hence, it has least and greatest fixpoints in this set. We define $\Gamma_I^{\mathcal{D}}(J)$ inductively as the interpretation $K + K'$ where

- K is the $(Def(\mathcal{D}) \setminus Def(\mathcal{R}))$-interpretation such that, for $J' = I + J|_{Def(\mathcal{R})}$:
  - $K|_{Def(\Delta_i)} = lfp(\Gamma_{J'}^{\Delta_i})$
  - $K|_{Def(\nabla_j)} = gfp(\Gamma_{J'}^{\nabla_j})$

Observe that J' interprets all open symbols in every subdefinition of $\mathcal{D}$.

- K' is the $Def(\mathcal{R})$-interpretation $\Gamma^{\mathcal{R}}_{I+K}(J|_{Def(\mathcal{R})})$.

Definition 5.6.5 (LFD model extending I_O) *Let $\mathcal{D}$ be a least fixpoint definition over Σ and I_O an $Open(\mathcal{D})$-interpretation interpreting all open symbols of $\mathcal{D}$ and no defined symbols of $\mathcal{D}$. The* model of $\mathcal{D}$ extending I_O*, denoted $lfm_{\mathcal{D}}(I_O)$, is the limit of the sequence $\langle I^i\rangle_{0\leq i<\xi}$ of Σ-interpretations, such that $I^0 = I_O[S/\mathbf{f}]$, where $S = \{P(\bar{d}) \mid P \in Def(\mathcal{R})$ and $\bar{d}$ is an arbitrary tuple of domain elements$\}$, and each I^{i+1} is derived from I^i by $I^{i+1} = \Gamma^{\mathcal{D}}_{I_O}(I^i)$.*

Definition 5.6.6 (GFD model extending I_O) *Let $\mathcal{D}$ be a greatest fixpoint definition over Σ and I_O an $Open(\mathcal{D})$-interpretation interpreting all open symbols of $\mathcal{D}$ and no defined symbols of $\mathcal{D}$. The* model of $\mathcal{D}$ extending I_O*, denoted $gfm_{\mathcal{D}}(I_O)$, is the limit of the sequence $\langle I^i\rangle_{0\leq i<\xi}$ of Σ-interpretations, such that $I^0 = I_O[S/\mathbf{t}]$, where $S = \{P(\bar{d}) \mid P \in Def(\mathcal{R})$ and $\bar{d}$ is an arbitrary tuple of domain elements$\}$, and each I^{i+1} is derived from I^i by $I^{i+1} = \Gamma^{\mathcal{D}}_{I_O}(I^i)$.*

Notice that $\mathrm{lfm}_{\mathcal{D}}(I_O)|_{Def(\mathcal{D})} = lfp(\Gamma^{\mathcal{D}}_{I_O})$ and $\mathrm{gfm}_{\mathcal{D}}(I_O)|_{Def(\mathcal{D})} = gfp(\Gamma^{\mathcal{D}}_{I_O})$.

Definition 5.6.7 (Model of $\mathcal{D}$) *Let $\mathcal{D}$ be a fixpoint definition and I a two-valued Σ-interpretation such that Σ contains all symbols in $\mathcal{D}$.*

- *If $\mathcal{D}$ is an LFD, then I* satisfies *$\mathcal{D}$, or I is a* model *of $\mathcal{D}$, denoted $I \models \mathcal{D}$, iff $I|_{Def(\mathcal{D})} = lfp(\Gamma^{\mathcal{D}}_{I|_{Open(\mathcal{D})}})$.*
- *If $\mathcal{D}$ is a GFD, then I* satisfies *$\mathcal{D}$, or I is a* model *of $\mathcal{D}$, denoted $I \models \mathcal{D}$, iff $I|_{Def(\mathcal{D})} = gfp(\Gamma^{\mathcal{D}}_{I|_{Open(\mathcal{D})}})$.*

Definition 5.6.8 (Model of an FO(FD) theory) *Let T be an FO(FD) theory over Σ and I a two-valued Σ-interpretation. Then I is a* model *of T, denoted $I \models T$, iff $I \models \varphi$ for every FO sentence or fixpoint definition $\varphi \in T$.*

Example 5.6.9 *In Example 5.6.4, the i'th iteration of the operator computes P as the set of states on a path leading to a sequence of at least i states satisfying R. Thus, the fixpoint consists of nodes with a path containing infinitely many states satisfying R.*

Three-valued semantics

The purpose of the propositional model generation techniques is to *construct* a two-valued model. However, during this construction, the eventual truth value of

some atoms is still unknown—initially, *every* truth value is unknown. We extend the semantics of FO(FD) as given above to three-valued semantics.

Let $\mathcal{R}$ be a set of rules over Σ and I a three-valued Σ-interpretation. For three-valued Σ-interpretations J, K such that $dom(J) = dom(K) = dom(I)$, we define $\Gamma_I^{\mathcal{R}}(J) = K$ if for each $P \in Def(\mathcal{R})$, $K(P) = \{\bar{d} | I|_{Open(\mathcal{R})} + J|_{Def(\mathcal{R})} \models \varphi_P[\bar{d}]\}$.

If each defined symbol in $Def(\mathcal{R})$ has only positive occurrences in the body of a rule in $\mathcal{R}$, the operator $\Gamma_I^{\mathcal{R}}$ is monotone with respect to the standard truth order on interpretations and hence, it has least and greatest fixpoints in this set denoted $lfp(\Gamma_I^{\mathcal{R}})$, respectively $gfp(\Gamma_I^{\mathcal{R}})$. Importantly, if $I(P) \leq I'(P)$ for every symbol $P \in Open(\mathcal{R})$ with only positive occurrences in rule bodies of $\mathcal{R}$, then $lfp(\Gamma_I^{\mathcal{R}}) \leq lfp(\Gamma_{I'}^{\mathcal{R}})$ and $gfp(\Gamma_I^{\mathcal{R}}) \leq gfp(\Gamma_{I'}^{\mathcal{R}})$.

Given an expression $\mathcal{D}$ over Σ which might be an LFD or a GFD, and a three-valued Σ-interpretation I. We define an operator $\Gamma_I^{\mathcal{D}}$ on the set of three-valued Σ-interpretations with domain $dom(I)$. This operator is monotone with respect to the standard truth order on interpretations and hence, it has least and greatest fixpoints in this set. We define $\Gamma_I^{\mathcal{D}}(J)$ inductively as the interpretation $I|_{Open(\mathcal{D})} + K|_{Def(\mathcal{D}) \setminus Def(\mathcal{R})} + K'|_{Def(\mathcal{R})}$ where

- for $J' = I|_{Open(\mathcal{D})} + J|_{Def(\mathcal{D})}$:
  - $K|_{Def(\Delta_i)} = lfp(\Gamma_{J'}^{\Delta_i})$
  - $K|_{Def(\nabla_j)} = gfp(\Gamma_{J'}^{\nabla_j})$
- $K'|_{Def(\mathcal{R})} = \Gamma^{\mathcal{R}}_{I|_{Open(\mathcal{D})} + K|_{Def(\mathcal{D})}}(J)$.

Definition 5.6.10 (Three-valued LFD model extending I) *Let $\mathcal{D}$ be a least fixpoint definition over Σ and I a three-valued Σ-interpretation. The* three-valued model of $\mathcal{D}$ extending I *is the limit of the sequence $\langle I^i \rangle_{0 \leq i < \xi}$ of three-valued Σ-interpretations, such that $I^0 = I[S/\mathbf{f}]$, where $S = \{P(\bar{d}) \mid P \in Def(\mathcal{R})$ and $\bar{d}$ is an arbitrary tuple of domain elements$\}$, and each I^{i+1} is derived from I^i by $I^{i+1} = \Gamma_I^{\mathcal{D}}(I^i)$.*

Definition 5.6.11 (Three-valued GFD model extending I) *Let $\mathcal{D}$ be a greatest fixpoint definition over Σ and I a three-valued Σ-interpretation. The* three-valued model of $\mathcal{D}$ extending I *is the limit of the sequence $\langle I^i \rangle_{0 \leq i < \xi}$ of three-valued Σ-interpretations, such that $I^0 = I[S/\mathbf{t}]$, where $S = \{P(\bar{d}) \mid P \in Def(\mathcal{R})$ and $\bar{d}$ is an arbitrary tuple of domain elements$\}$, and each I^{i+1} is derived from I^i by $I^{i+1} = \Gamma_I^{\mathcal{D}}(I^i)$.*

It is worth mentioning that if a Σ-interpretation I is three-valued but $I|_{Open(\mathcal{D})}$ is two-valued, then the model of $\mathcal{D}$ extending I is two-valued.

Definition 5.6.12 (Three-valued model of $\mathcal{D}$) *Let $\mathcal{D}$ be a fixpoint definition and I a three-valued Σ-interpretation such that Σ contains all symbols in $\mathcal{D}$.*

- *If $\mathcal{D}$ is an LFD, then I is a* three-valued model *of $\mathcal{D}$ iff $I|_{Def(\mathcal{D})} = lfp(\Gamma^{\mathcal{D}}_{I|_{Open(\mathcal{D})}})$.*
- *If $\mathcal{D}$ is a GFD, then I is a* three-valued model *of $\mathcal{D}$ iff $I|_{Def(\mathcal{D})} = gfp(\Gamma^{\mathcal{D}}_{I|_{Open(\mathcal{D})}})$.*

PC(FD) and SAT(FD)

The Boolean *satisfiability problem* (SAT) is the problem of deciding whether a given PC theory has a model. A famous result by Cook [41] states that the SAT problem is NP-complete.

Decision procedure for SAT were developed by Davis and Putnam [44] and Davis et al. [43]; the procedure proposed in the latter paper is now commonly referred to as "the Davis-Putnam-Logemann-Loveland (DPLL) procedure". Early implementations of this and other procedures were not widely successful. This changed however when *clause learning* was added to the DPLL procedure [129]. Nowadays, SAT solving has many practical applications in industry and is a very active research domain [30].

Definition 5.6.13 (PC(FD), SAT(FD)) *The propositional fragment of FO(FD) is called* PC(FD). *The satisfiability problem of PC(FD) is called* SAT(FD).

A propositional fixpoint definition is a fixpoint definition such that all symbols occurring in it are propositional symbols. Observe that PC(FD) is an extension of PC with propositional fixpoint definitions. A *PC(FD) theory* is a set of PC formulas and propositional fixpoint definitions.

As we will see in Chapter 6, the SAT(FD) problem of a given PC(FD) theory is solved in practice by searching for a model of that theory. Hence, if we can solve the SAT(FD) problem, we can solve the model generation problem for PC(FD), i.e., the *propositional* model generation problem for FO(FD).

Remark 5.6.14 *The semantics of propositional fixpoint definitions is just derived from that on the first-order level. Therefore, the intended semantics of propositional fixpoint definitions is derived from the intended semantics of first-order fixpoint definitions.*

Example 5.6.15 *Consider the propositional fixpoint definition:*

$$\mathcal{D} = \left\{ \begin{array}{l} p \leftarrow q \vee r \\ q \leftarrow p \\ \left[\begin{array}{l} r \leftarrow p \\ s \leftarrow t \vee a \\ t \leftarrow s \end{array} \right] \end{array} \right\}.$$

It is obvious that a is the only open atom in this fixpoint definition. There are only two two-valued interpretations satisfying $\mathcal{D}$, namely, $I_1 = \{a \mapsto \mathbf{f}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{f}, s \mapsto \mathbf{t}, t \mapsto \mathbf{t}\}$ and $I_2 = \{a \mapsto \mathbf{t}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{f}, s \mapsto \mathbf{t}, t \mapsto \mathbf{t}\}$.

Give $I_O = \{a \mapsto \mathbf{f}\}$, the construction of $gfm_{\mathcal{D}}(I_O)$ is illustrated as follows: $I^0 = \{a \mapsto \mathbf{f}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}\}$, $I^1 = \{a \mapsto \mathbf{f}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{f}, s \mapsto \mathbf{t}, t \mapsto \mathbf{t}\}$, which is the limit of the iterations and thus, $gfm_{\mathcal{D}}(I_O) = I^1 = I_1$.

Given the three-valued interpretation $I = \{a \mapsto \mathbf{t}, p \mapsto \mathbf{u}, q \mapsto \mathbf{u}, r \mapsto \mathbf{u}, s \mapsto \mathbf{u}, t \mapsto \mathbf{u}\}$, the construction of the three-valued model of $\mathcal{D}$ extending I is illustrated as follows: $I^0 = \{a \mapsto \mathbf{t}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{u}, s \mapsto \mathbf{u}, r \mapsto \mathbf{u}\}$, $I^1 = \{a \mapsto \mathbf{t}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{f}, s \mapsto \mathbf{t}, t \mapsto \mathbf{t}\}$, which is the limit of the iterations and therefore, it is the model of $\mathcal{D}$ extending I. Note that this model is two-valued due to the fact that $I|_{Open(\mathcal{D})}$ is two-valued.

5.7 A comparison of FO(FD) and FO(ID)

In this section, we illustrate the relationship between FO(FD) and FO(ID). We present equivalence preserving transformations from non-monotone inductive definitions in FO(ID) to alternating fixpoint definitions. These transformations may introduce new symbols to the original vocabulary Σ: thus the type of equivalence we obtain is *up to* Σ. We also compare FO(FD) with FO(ID) in terms of expressive power.

Recall that a (generalized) inductive definition in FO(ID) is a finite set of rules. We do not insist on defined predicates to occur positively in rule bodies in a generalized inductive definition, meaning that non-monotone inductive definitions are allowed.

Example 5.7.1 *Consider the following generalized inductive definition of even and odd numbers over the structure of the natural numbers with zero and the successor function:*

$$\left\{ \begin{array}{l} \forall x(Even(x) \leftarrow x = 0 \vee \exists y(x = s(y) \wedge \neg Even(y))) \\ \forall x(Odd(x) \leftarrow \exists y(x = s(y) \wedge Even(y))) \end{array} \right\}.$$

Definition 5.7.2 *Let Σ be a vocabulary, D a generalized inductive definition over Σ and $\mathcal{I}_O$ an Open(D)-interpretation. Denote by L_D the set of all Σ-interpretations extending $\mathcal{I}_O$ and define the operator $\Phi_D : L_D \to L_D$ by $\mathcal{I}' = \Phi_D(\mathcal{I})$ iff*

- *$dom(\mathcal{I}') = dom(\mathcal{I})$,*
- *for each open symbol O, $O^{\mathcal{I}'} = O^{\mathcal{I}}$ and*
- *for each defined symbol $P \in Def(D)$,*

$$P^{\mathcal{I}'} = \{\bar{d} \mid \mathcal{I} \models \Psi_P[\bar{d}]\}.$$

In general, Φ_D is a non-monotone operator. If D is a positive GID, Φ_D is monotone, and the *least model of D extending $\mathcal{I}_O$* is defined as the least fixpoint of Φ_D.

The negative occurrences of defined symbols in the rule bodies are responsible for the non-monotone behaviour of the operator Φ_D: adding more tuples to the value of a negatively occurring defined symbol in $\Psi_P[\bar{x}]$ has an anti-monotone effect on the derived relation and may lead to the derivation of fewer tuples $\bar{d}$ satisfying this formula. Thus we can eliminate the non-monotonicity of Φ_D and set up a monotone induction process using D if we *fix* the value of negative occurrences of defined symbols in rule bodies. Suppose we choose a fixed interpretation $\mathcal{I}$ to evaluate the negative occurrences of defined symbols in rule bodies. We can then perform a monotonic derivation process $\bot_{\mathcal{I}|_{Open(D)}}, \mathcal{K}^1, \mathcal{K}^2, \ldots$ in which each $\mathcal{K}^{i+1}$ is derived from D by evaluating positive occurrences of defined symbols in each $\Psi_P(\bar{x})$ with respect to $\mathcal{K}^i$ and negative occurrences with respect to $\mathcal{I}$. This process will be monotone.

Let D be a generalized inductive definition over Σ and $\mathcal{I}$ a Σ-interpretation. The *reduct* of D in $\mathcal{I}$, denoted by $D^{\mathcal{I}}$, is obtained by fixing the value of all negative occurrences of defined domain atoms $P(\bar{d})$ in every rule as $\mathcal{I}(P(\bar{d}))$.

Definition 5.7.3 (Stable model) *Let D be a generalized inductive definition over Σ, $\mathcal{I}_O$ an Open(D)-interpretation. Then a Σ-interpretation $\mathcal{I}$ is a* stable model of D extending $\mathcal{I}_O$ *iff $\mathcal{I}|_{Def(D)}$ is the least model of $D^{\mathcal{I}}$ extending $\mathcal{I}_O$ and $\mathcal{I}|_{Open(D)} = \mathcal{I}_O$.*

By Proposition 2.3.17, we know that if the well-founded model $\mathcal{I}$ of D extending $\mathcal{I}_O$ is two-valued, then $\mathcal{I}$ is also the unique stable model of D extending $\mathcal{I}_O$.

We now define an equivalence preserving transformation from an arbitrary inductive definition to an alternating fixpoint definition.

Definition 5.7.4 *Let D be a generalized inductive definition. For each defined predicate P of D, we introduce two new predicate symbols P^c and $P^{c\neg}$ of the same arity of P. For each formula φ, let $\overline{\varphi}$ denote the formula obtained by substituting each positive occurrence $P(\bar{t})$ of a defined predicate P in φ by $P^c(\bar{t})$ and each negative occurrence $P(\bar{t})$ in φ by $\neg P^{c\neg}(\bar{t})$. We define two sets of rules:*

$$\mathcal{R}^c_D = \{\forall \bar{x}(P^c(\bar{x}) \leftarrow \overline{\Psi_P[\bar{x}]}) \mid P \in Def(D)\},$$

$$\mathcal{R}^{c\neg}_D = \{\forall \bar{x}(P^{c\neg}(\bar{x}) \leftarrow \overline{\neg\Psi_P[\bar{x}]}) \mid P \in Def(D)\}.$$

Now define Δ_D as

$$\lfloor\ \mathcal{R}^c_D, \lceil\ \mathcal{R}^{c\neg}_D\ \rceil\ \rfloor.$$

Let D be a generalized inductive definition over Σ. Then Δ_D is a least fixpoint definition over $\Sigma' = \Sigma \cup \{P^c \mid P \in Def(D)\} \cup \{P^{c\neg} \mid P \in Def(D)\}$. Note that $Open(D) = Open(\Delta_D)$.

Example 5.7.5 *Let D be*

$$\{\ P \leftarrow P \wedge Q\ \}.$$

We have $\Delta_D =$

$$\left\lfloor \begin{array}{l} P^c \leftarrow P^c \wedge Q \\ \lceil\ P^{c\neg} \leftarrow P^{c\neg} \vee \neg Q\ \rceil \end{array} \right\rfloor.$$

Example 5.7.6 *Let D be*

$$\left\{ \begin{array}{l} P \leftarrow P \vee \neg Q \\ Q \leftarrow Q \vee \neg P \end{array} \right\}.$$

We have $\Delta_D =$

$$\left\lfloor \begin{array}{l} P^c \leftarrow P^c \vee Q^{c\neg} \\ Q^c \leftarrow Q^c \vee P^{c\neg} \\ \left\lceil \begin{array}{l} P^{c\neg} \leftarrow P^{c\neg} \wedge Q^c \\ Q^{c\neg} \leftarrow Q^{c\neg} \wedge P^c \end{array} \right\rceil \end{array} \right\rfloor.$$

Example 5.7.7 *Let $D =$*

$$\left\{ \begin{array}{l} \forall x(Even(x) \leftarrow x = 0 \vee \exists y(x = s(y) \wedge \neg Even(y))) \\ \forall x(Odd(x) \leftarrow \exists y(x = s(y) \wedge Even(y))) \end{array} \right\}.$$

We have $\Delta_D =$

$$\left\lfloor \begin{array}{l} \forall x(Even^c(x) \leftarrow x = 0 \vee \exists y(x = s(y) \wedge Even^{c\neg}(y))) \\ \forall x(Odd^c(x) \leftarrow \exists y(x = s(y) \wedge Even^c(y))) \\ \left\lceil \begin{array}{l} \forall x(Even^{c\neg}(x) \leftarrow x \neq 0 \wedge \forall y(x = s(y) \supset Even^c(y))) \\ \forall x(Odd^{c\neg}(x) \leftarrow \forall y(x = s(y) \supset Even^{c\neg}(y))) \end{array} \right\rceil \end{array} \right\rfloor.$$

We now show that transformations from generalized inductive definitions to alternating fixpoint definitions preserve equivalence.

Theorem 5.7.8 *Let D be a generalized inductive definition over Σ. Then there exists a one-to-one mapping between the Σ-models $\mathcal{I}$ of D and the Σ'-models $\mathcal{I}'$ of Δ_D with $dom(\mathcal{I}) = dom(\mathcal{I}')$, $\mathcal{I}'|_\Sigma = \mathcal{I}$, $\mathcal{I}'(P^c) = \mathcal{I}(P)$ and $\mathcal{I}'(P^{c\neg}) = \mathcal{I}(\neg P)$ for each $P \in Def(D)$.*

Proof. Denote by $\mathcal{I}^1$ the Σ-interpretation such that $\mathcal{I}^1(O) = \mathcal{I}_O(O)$ for every $O \in Open(D)$ and $\mathcal{I}^1(P(\bar{d})) = \mathbf{f}$ for each defined domain atom $P(\bar{d})$ of D. Let $\mathcal{I}^{n+1} = lfp(\Gamma^{D^{\mathcal{I}^n}}_{\mathcal{I}^n|_{Open(D)}})$ for every $n \geq 1$. It is easy to show that the limit of $(\mathcal{I}_n)_{n\geq 1}$, denoted by $\mathcal{I}$, is a stable model of D extending I_O. Since D has a two-valued well-founded model extending I_O, we have that $\mathcal{I}$ is the two-valued well-founded model of D extending I_O by Proposition 2.3.17.

Denote by $\mathcal{J}^1$ the Σ'-interpretation such that Σ' is the approximation vocabulary of Σ with respect to D, $\mathcal{J}^1(O) = \mathcal{I}_O(O)$ for every $O \in Open(\Delta_D)$, $\mathcal{J}^1(P^c(\bar{d})) = \mathbf{f}$ for each defined domain atom $P^c(\bar{d})$ of $\mathcal{R}^c_D$, and $\mathcal{J}^1(P^{c\neg}(\bar{d})) = \mathbf{t}$ for each defined domain atom $P^{c\neg}(\bar{d})$ of $\mathcal{R}^{c\neg}_D$. Let $\mathcal{J}^{n+1}|_{Def(\mathcal{R}^{c\neg}_D)} = gfp(\Gamma^{\mathcal{R}^{c\neg}_D}_{\mathcal{J}^n|_{Open(\Delta_D)}+\mathcal{J}^n|_{Def(\mathcal{R}^c_D)}})$ and $\mathcal{J}^{n+1}|_{Def(\mathcal{R}^c_D)} = lfp(\Gamma^{\mathcal{R}^c_D}_{\mathcal{J}^n|_{Open(\Delta_D)}+\mathcal{J}^n|_{Def(\mathcal{R}^{c\neg}_D)}})$ for every $n \geq 1$, and finally denote the limit of $(\mathcal{J}^n)_{n\geq 1}$ by $\mathcal{J}$. It can be easily shown that $\mathcal{J}$ is a model of Δ_D extending I_O. In the following, we want to show by induction that for every $n \geq 1$, $\mathcal{I}^n(P) = \mathcal{J}^n(P^c)$ for every $P \in Def(D)$ and $\mathcal{J}^n(P^{c\neg}(\bar{d})) = \neg\mathcal{I}^n(P(\bar{d}))$ for each defined domain atom $P(\bar{d})$ of D, and thus $\mathcal{J}$ encodes $\mathcal{I}$, as desired.

- (base case) $n = 1$, trivially.
- (induction case) By the construction of $\mathcal{R}^c_D$ and the induction hypothesis that $\mathcal{I}^n(P) = \mathcal{J}^n(P^c)$ for every $P \in Def(D)$ and $\mathcal{J}^n(P^{c\neg}(\bar{d})) = \neg\mathcal{I}^n(P(\bar{d}))$ for each defined domain atom $P(\bar{d})$ of D, we have that $\mathcal{J}^{n+1}(P^c) = lfp(\Gamma^{\mathcal{R}^c_D}_{\mathcal{J}^n|_{Open(\Delta_D)}+\mathcal{J}^n|_{Def(\mathcal{R}^{c\neg}_D)}})(P^c) = lfp(\Gamma^{D^{\mathcal{I}^n}}_{\mathcal{I}^n|_{Open(D)}})(P) = \mathcal{I}^{n+1}(P)$ for each $P \in Def(D)$. Consider the following Σ'-interpretation $\mathcal{K}$ such that $\mathcal{K}|_{Def(\mathcal{R}^{c\neg}_D)} = lfp(\Gamma^{\mathcal{R}^{c\neg\neg}_D}_{\mathcal{J}^n|_{Open(\Delta_D)}+\mathcal{J}^n|_{Def(\mathcal{R}^c_D)}})$ where $\mathcal{R}^{c\neg\neg}_D = \{\forall\bar{x}(P^{c\neg}(\bar{x}) \leftarrow \neg(\overline{\neg\Psi_P[\bar{x}]}[P^{c\neg}/\neg P^{c\neg}])) \mid P \in Def(D)\}$ and $\overline{\neg\Psi_P[\bar{x}]}[P^{c\neg}/\neg P^{c\neg}]$ is obtained by substituting each positive occurrence of $P^{c\neg}(\bar{t})$ in $\overline{\neg\Psi_P[\bar{x}]}$ by $\neg P^{c\neg}(\bar{t})$ for each $P^{c\neg} \in Def(\mathcal{R}^{c\neg}_D)$. It can be obtained that $\mathcal{J}^{n+1}(P^{c\neg}(\bar{d})) = \neg\mathcal{K}(P^{c\neg}(\bar{d}))$ for every defined domain atom $P^{c\neg}(\bar{d})$ of $\mathcal{R}^{c\neg}_D$. By the construction of $\mathcal{R}^{c\neg\neg}_D$ and the induction hypothesis that $\mathcal{I}^n(P) = \mathcal{J}^n(P^c)$ for every $P \in Def(D)$ and $\mathcal{J}^n(P^{c\neg}(\bar{d})) = \neg\mathcal{I}^n(P(\bar{d}))$ for each defined domain atom $P(\bar{d})$ of D, we have that $\mathcal{K}(P^{c\neg}(\bar{d})) = \mathcal{I}^{n+1}(P(\bar{d})))$ for each defined domain atom $P(\bar{d})$ of

D, and hence that $\mathcal{J}^{n+1}(P^{c\neg}(\bar{d})) = \neg\mathcal{I}^{n+1}(P(\bar{d}))$ for each defined domain atom $P(\bar{d})$ of D.

In the following we show that FO(FD) and FO(ID) do not have the same expressive power in the context of infinite structures.

Theorem 4.4 in [155], for the well-founded semantics, states that a relation is definable in the well-founded semantics if and only if it is inductively (Π^1_1) definable over the natural numbers. However, on the other hand, Theorem 10 in [24] presents that the FO(FD) alternation hierarchy, which is the hierarchy of alternating LFD and GFD expressions (ordered along the number of alternations) in any fixpoint definitions, is strict. This discussion leads to the following result:

Theorem 5.7.9 *FO(FD) is strictly more expressive than FO(ID) on infinite structures.*

5.8 Coinductive logic programming vs. FO(FD)

Coinduction has been incorporated in logic programming in a systematic way only recently [164, 165, 81], where an operational semantics—similar to SLD [6]—is given for computing the greatest fixpoint of a logic program. A coinductive logic program is syntactically identical to a traditional logic program, i.e., a coinductive logic program is a finite set of rules, except that it indicates which predicates are coinductively defined at the beginning of the program. Syntactic terms and atoms have their traditional inductive definitions. However, coinductive LP also makes use of infinite terms and atoms in its semantics. These generalizations of syntactic terms and atoms have a straightforward definition as the greatest fixpoints of the same respective generating functions used to define syntactic terms and atoms. We illustrate coinductive logic programming via the following example.

Example 5.8.1 (Infinite streams) *This example involves a combination of an inductive predicate and a coinductive predicate. By default, predicates are inductive, unless indicated otherwise. Consider the execution of the following program, which defines a predicate that recognizes infinite streams of natural numbers. Note that only the* stream/1 *predicate is coinductive, while the* number/1 *predicate is inductive, and* s *is the successor function on the natural numbers.*

$$\left\{\begin{array}{c} \mathsf{coinductive\ stream/1} \\ \mathsf{stream([H|T])} \leftarrow \mathsf{number(H)} \wedge \mathsf{stream(T)} \\ \mathsf{number(0)} \\ \mathsf{number(s(N))} \leftarrow \mathsf{number(N)} \\ \mathsf{stream([0,\ s(0),\ s(s(0))\ |\ T])} \leftarrow \end{array}\right\}$$

The declarative semantics of a coinductive logic program is the "across the board" dual of the traditional minimal model Herbrand semantics, i.e., the model of a coinductive logic program P, written $M^{co}(P)$, is the *maximal infinitary Herbrand model* (or, *co-Herbrand model*) of a program P. The maximal infinitary Herbrand model is the greatest fixpoint of $\mathcal{T}_P$ with atoms ranging over the *infinitary Herbrand base* (or, *co-Herbrand base*). The details can be found in [89, 109, 135]. As demonstrated in [89, 109, 135], this allows the universe of terms, called the *infinitary Herbrand universe* (or, *co-Herbrand universe*), to contain infinite terms, in addition to the traditional finite terms, and similarly for the infinitary Herbrand base, which contains finite and infinite atoms. It also allows for the model to contain ground goals that have either finite or infinite proofs.

Logic programming also has a goal directed, top-down operational semantics, called SLD, which stands for "**S**election function with **L**inear resolution for **D**efinite programs". SLD can be thought of as a non-deterministic state transition system corresponding to the high-level state transitions of an execution of a logic programming query. The operational semantics given for coinductive LP is defined in a manner similar to SLD, and is therefore called *co-SLD*. Where SLD uses sets of syntactic atoms and syntactic term substitutions for states, co-SLD uses finite trees of syntactic atoms along with systems of equations. Of course, the traditional goals of SLD can be extracted from these trees, as the goal of a tree is simply the set of leaves of the tree. Furthermore, where SLD only allows program rules as state transition rules, co-SLD also allows an implicit *coinductive hypothesis rule* for providing atoms that have an infinite proof, with a finite derivation. As is the case with SLD, it is up to the underlying search strategy to find a sequence of transition rules that prove the original query.

It is also discussed in [81] that coinductive logic programming has applications to verification and model checking, lazy evaluation, concurrent logic programming and non-monotonic reasoning.

FO(FD) is an integration of classical logic with mixed induction and coinduction. A Logic Programming (LP-like) rule notation is used for expressing inductive and coinductive concepts in FO(FD). The semantics of FO(FD) is an integration of standard FO semantics with fixpoint semantics. Therefore, coinductive logic programming and FO(FD) are analogous to each other in both syntactic and semantic ways. However, FO(FD) is more general in syntax than coinductive logic programming since naively mixing induction and coinduction in coinductive logic programming leads to contradictions, while arbitrary cyclical nesting of least and greatest fixpoint expressions is allowed in FO(FD). Consider the following

coinductive program where the predicate p is coinductive and q is inductive:

$$\left\{ \begin{array}{c} \text{coinductive } p \\ p \leftarrow q \\ q \leftarrow p \end{array} \right\}.$$

For computing the result of goal $\neg q$, we will use least fixpoint semantics, which will produce null, implying that q should fail, and p should also fail, since p calls q. However, if we use greatest fixpoint semantics (and the coinductive hypothesis computation rule), the goal p should succeed, which, in turn, implies that q should succeed. This contradiction is resolved by disallowing such cyclical nesting of inductive and coinductive predicates, i.e., stratifying inductive and coinductive predicates in a program. An inductive predicate in a given strata cannot call a coinductive predicate defined in a higher strata and vice versa [81, 165].

However, such kind of contradictions do not exist in FO(FD) since initially a fixpoint definition is declared to be either a least or a greatest one. Consider the same example. p, respectively q, should be locally defined in a greatest fixpoint definition, respectively a least fixpoint definition. Consider the least fixpoint definition: $\left\lfloor \begin{array}{l} q \leftarrow p \\ \lceil\ p \leftarrow q\ \rceil \end{array} \right\rfloor$. Both q and p are derived to be false. We can also derive that both p and q are true by changing this fixpoint definition to be a greatest one as: $\left\lceil \begin{array}{l} p \leftarrow q \\ \lfloor\ q \leftarrow p\ \rfloor \end{array} \right\rceil$. So arbitrary mixing of least and greatest fixpoint expressions in a fixpoint definition is allowed in FO(FD), and it follows that FO(FD) is more general in syntax than coinductive logic programming.

Another difference between FO(FD) and coinductive logic programming is on the computational level. The main computational task for FO(FD) is to make model generation in the context of a finite domain. However, the model generation in coinductive logic programming is applied to construct an infinite Herbrand model based on an infinite Herbrand universe.

5.9 Conclusion

In this chapter, we have introduced induction and coinduction in detail. The brief history of fixpoint theory in Computer Science has been investigated. We have also illustrated the relationship between various fixpoint logics and FO(ID).

We have introduced fixpoint definitions, which is an alternative expression of fixpoint constructs, and the logic FO(FD), which is an extension of first-order logic with fixpoint definitions. We have investigated the correspondence between fixpoint definitions and non-monotone inductive definitions in FO(ID) by providing

equivalent transformations from non-monotonic inductive definitions in FO(ID) to alternating fixpoint definitions. We have compared the expressive power of FO(FD) and FO(ID) in the context of infinite structures.

Related work is provided by Gupta et al. in [81, 165, 164]. They introduced coinduction, which corresponds to the greatest fixpoint constructor, into logic programming to obtain coinductive logic programming. They discussed applications of coinductive logic programming into programming verification, model checking and non-monotonic reasoning, in particular its manifestation as answer set programming. We have compared FO(FD) and coinductive logic programming as well.

Chapter 6

SAT(FD)

6.1 Introduction

The SAT problem, deciding the satisfiability of propositional logic theories, is a major research theme. The research domain of SAT has thrived in recent years; many highly efficient "SAT solvers" have been developed (see e.g., [129]). An important research direction is to develop SAT solvers for extensions of PC (e.g., [138]). The use of extended languages leads to broader applicability of SAT-like systems, facilitates the modelling of applications, and may substantially reduce the size of encodings. All these benefits also hold for PC(FD). This inspires us to extend SAT solvers with techniques to handle fixpoint definitions in this chapter.

Section 6.2 introduces a normal form for a PC(ID) theory, which we use throughout the text, and present some graph concepts, which will be used in Section 6.4.

We present a standard algorithm, which was proposed in [27], for the evalaution of fixpoint expressions in the form of μ-calculus in Section 6.3. This algorithm makes extensive use of monotonicity consideration to solve the fixpoint computation over a finite lattice in about $n^{2/d}$ steps, where n is the maximum length of a chain in the lattice and d is the number of alternations between least and greatest fixpoints. We develop an algorithm for the computation of propositional fixpoint definitions using ideas of Browne et al.'s algorithm. We provide a complexity result for the SAT(FD) problem.

We give a semantical characterization of (models of) FDs in Section 6.4. This one is a graph theoretical characterization of FDs; it uses graph structures called *justifications*. Such a semantical analysis is important because it yields new

insights into the logic. This result also forms the theoretical underpinnings to develop algorithms for solving the SAT(FD) problem.

We present a high-level overview of SAT(FD) in Section 6.5. Our approach to the SAT(FD) problem is by supporting FDs in SAT solvers. We present how SAT(FD) solvers can be built by extending SAT solvers with an additional propagation mechanism suitable for reasoning on fixpoint definitions. We then develop various algorithms that can be used to implement a SAT(FD) solver in Section 6.6.

We implemented a prototype SAT(FD) solver MiniSat(FD), which we present in Section 6.7. MiniSat(FD) is built as an extension of the SAT solver MiniSat [60]. We introduce a computational paradigm of *model expansion*, which is a generalization of finite model generation. We then present a concrete system to solve model expansion problems for FO(FD): the MX(FO(FD)) system. Such a system consists of two components: a grounder and a propositional model generator, MiniSat(FD). We also evaluate the performance of MX(FO(FD)) by a benchmark borrowing from the model checking domain. Finally, in Section 6.8 we conclude and discuss related work.

6.2 Preliminaries

6.2.1 Completion

Clark [34] defined the *completion* of a logic program. Here we similarly define the completion of a propositional fixpoint definition (FD) $\mathcal{D}$, denoted by $comp(\mathcal{D})$: it is the propositional formula

$$\bigwedge_{p \in Def(\mathcal{D})} (p \equiv \bigvee \{\varphi \mid (p \leftarrow \varphi) \in \mathcal{D}\}).$$

For a PC(FD) theory T, denote by $comp(T)$ the theory obtained from T by replacing each definition $\mathcal{D} \in T$ by $comp(\mathcal{D})$.

The following result is well-known:

Proposition 6.2.1 *Let $\mathcal{D}$ be a propositional fixpoint definition. Then $\mathcal{D} \models comp(\mathcal{D})$.*

Note that for any given $Open(\mathcal{D})$-interpretation I_O, $\mathcal{D}$ has a unique model extending I_O. $comp(\mathcal{D})$ on the other hand may have multiple models extending I_O.

Example 6.2.2 *Let* $\mathcal{D} = \left\lfloor \begin{array}{l} p \leftarrow p \vee a \\ \lceil\ q \leftarrow q \wedge p\ \rceil \end{array} \right\rfloor$.

Then $comp(\mathcal{D}) = (p \equiv p \vee a) \wedge (q \equiv q \wedge p)$. $\mathcal{D}$ *has two models:* $\{a \mapsto \mathbf{f}, p \mapsto \mathbf{f}, q \mapsto \mathbf{f}\}$ *and* $\{a \mapsto \mathbf{t}, p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$*;* $comp(\mathcal{D})$ *has the same two models, and the additional three models:* $\{a \mapsto \mathbf{f}, p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$, $\{a \mapsto \mathbf{f}, p \mapsto \mathbf{t}, q \mapsto \mathbf{f}\}$ *and* $\{a \mapsto \mathbf{t}, p \mapsto \mathbf{t}, q \mapsto \mathbf{f}\}$.

6.2.2 Normal form

Recall that for a set of literals S, we denote by $\neg S$ the set $\{\neg s \mid s \in S\}$, and by $\widehat{S}$ the set $S \cup \neg S$.

A propositional fixpoint definition $\mathcal{D}$ is in *definitional normal form* (*DefNF*) if each $p \in Def(\mathcal{D})$ is defined by exactly one rule, denoted $p \leftarrow \varphi_p$, the body of which is either a disjunction or a conjunction of literals. We call $p \leftarrow \varphi_p$ the *rule defining* p. Slightly abusing notation, in some contexts φ_p will be used to denote the *set of literals* in the body of the rule defining p. E.g., $l \in \varphi_p$ means that l is one of the conjuncts or disjuncts of φ_p. We extend this notation to negative defined literals: for $l \in \neg Def(\mathcal{D})$, we denote by φ_l the formula $\neg\varphi_{\neg l}$, which is a conjunction if $\varphi_{\neg l}$ is a disjunction, and vice versa. Then the literals $l' \in \varphi_l$ are the negations of the conjuncts or disjuncts of $\varphi_{\neg l}$.

Observe that for DefNF definitions $\mathcal{D}$ the completion simplifies to

$$comp(\mathcal{D}) = \bigwedge_{p \in Def(\mathcal{D})} p \equiv \varphi_p,$$

i.e., a simple replacement of $\leftarrow$ by $\equiv$. It then follows from Proposition 6.2.1 that for any $p \in Def(\mathcal{D})$, $\mathcal{D} \models (p \equiv \varphi_p)$.

We partition the set of defined literals $\widehat{Def(\mathcal{D})}$ in two:

$$\mathbf{D}_{\text{lits}} = \{p \mid \varphi_p \text{ is a disjunction}\} \cup \{\neg p \mid \varphi_p \text{ is a conjunction}\},$$
$$\mathbf{C}_{\text{lits}} = \{p \mid \varphi_p \text{ is a conjunction}\} \cup \{\neg p \mid \varphi_p \text{ is a disjunction}\}.$$

For atoms $p \in Def(\mathcal{D})$ for which φ_p is a singleton, we treat φ_p as a conjunction. Also, we interpret $\top$ and $\bot$ here as literals of $Open(\mathcal{D})$, with fixed truth values $\mathbf{t}$ respectively $\mathbf{f}$.

We call literals in $\mathbf{D}_{\text{lits}}$ *disjunctively defined*, and literals in $\mathbf{C}_{\text{lits}}$ *conjunctively defined*. Observe that from $\mathcal{D} \models (p \equiv \varphi_p)$ for defined atoms p we can derive $\mathcal{D} \models (\neg p \equiv \neg\varphi_p)$; for a negative literal $l = \neg p$ that is disjunctively defined, φ_p is a conjunction, so $\neg\varphi_p$ is a disjunction; for a conjunctively defined negative literal $l = \neg p$, $\neg\varphi_p$ is a conjunction. Hence, all disjunctively defined literals are equivalent

to the disjunction of their body literals, and all conjunctively defined literals are equivalent to the conjunction of their body literals.

Example 6.2.3 *Let* $\mathcal{D} =$

$$\left\lfloor \begin{array}{l} p \leftarrow q \vee a \\ \lceil \; q \leftarrow p \wedge \neg b \; \rceil \end{array} \right\rfloor .$$

Then we have $\mathbf{D}_{lits} = \{p, \neg q\}$, $\mathbf{C}_{lits} = \{\neg p, q\}$, *and* $comp(\mathcal{D}) = (p \equiv q \vee a) \wedge (q \equiv p \wedge \neg b)$. *For the disjunctively defined literals* p *and* $\neg q$, *we have* $\varphi_p = \bigvee\{q, a\}$ *and* $\varphi_{\neg q} = \bigvee\{\neg p, b\}$, *and consequently it holds that* $\mathcal{D} \models p \equiv \bigvee\{q, a\}$ *and* $\mathcal{D} \models \neg q \equiv \bigvee\{\neg p, b\}$.

An arbitrary propositional fixpoint definition can be transformed in linear time into an equivalent (up to original vocabulary) DefNF definition using *predicate introduction*. The transformation starts by merging all rules with the same head atom into one rule as follows: $p \leftarrow \varphi_1, \ldots, p \leftarrow \varphi_n$ becomes $p \leftarrow \varphi_1 \vee \ldots \vee \varphi_n$, for all defined atoms p. It then simply applied the following two steps until DefNF form is reached:

1. replace a positively occurring subformula ψ of the body of some rule defining p by a newly introduced atom, say p_ψ;
2. introduce a new rule $p_\psi \leftarrow \psi$ into $\mathcal{D}'$ in which p is locally defined, such that p_ψ is locally defined in $\mathcal{D}'$ as well.

Its correctness can be proven following the method introduced in [183].

As an example, we show how a fixpoint definition in a normal logic programming form is transformed to DefNF. Let the definition consists of rules $p_i \leftarrow \psi_{i,j}$, for $1 \leq i \leq n$ and for $1 \leq j \leq n_i$, where p_i is locally defined in a subdefinition $\mathcal{D}_i$ and the $\psi_{i,j}$ are conjunctions of literals. The DefNF definition obtained after our transformation consists of disjunctive rules of the form $p_i \leftarrow \bigvee_{1 \leq j \leq n_i} p_{i,j}$, for $1 \leq i \leq n$, and of conjunctive rules of the form $p_{i,j} \leftarrow \psi_{i,j}$, for $1 \leq i \leq n$ and for $1 \leq j \leq n_i$. The $p_{i,j}$ are new atoms. Both p_i and each $p_{i,j}$ for $1 \leq j \leq n_i$ are locally defined in $\mathcal{D}_i$.

Besides the obvious advantage of simplified theoretical results, the use of DefNF has also a practical advantage:

- It is very similar to the widely used conjunctive normal form (CNF); in fact, transforming $comp(\mathcal{D})$ to CNF is trivial. This means that in practical implementations of SAT(FD) solvers, the data structures that are used by SAT solvers to represent CNF clauses can be reused to represent DefNF rules.

We study the satisfaction relation $I \models \mathcal{D}$ for DefNF fixpoint definitions $\mathcal{D}$ with vocabulary Σ. The results generalize, with appropriate adaption of definitions, to arbitrary propositional fixpoint definitions.

We now also define a normal form for PC(FD) theories. A PC(FD) theory that has at most one definition, which is in DefNF, and a propositional logic part, which is in conjunctive normal form (CNF), is said to be in *Extended CNF (ECNF)* normal form. Throughout this chapter, we assume a propositional vocabulary Σ and an ECNF theory consisting of a CNF formula Ψ and a DefNF definition $\mathcal{D}$.

6.2.3 Some graph concepts

A (*directed*) *graph* is a tuple (V, E) of *vertices* V and *edges* E. The edges form a binary relation over vertices. A *path* from $v_1 \in V$ to $v_2 \in V$ is a non-empty sequence of edges $(v_1, v'), (v', v''), \ldots, (v^{(n)}, v_2)$, each of which is in E. For a graph $G = (V, E)$, we denote by $v \in G$ that $v \in V$ and by $(v_1, v_2) \in G$ that $(v_1, v_2) \in E$. For graphs $G = (V, E)$ and $G' = (V', E')$, G' is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$.

Given a graph $G = (V, E)$, a set of vertices $S \subseteq V$ is said to be *strongly connected* if for any $v_1, v_2 \in S$, the graph contains a path through S from v_1 to v_2. A *loop* of a graph is a non-empty, strongly connected set of vertices. For instance, if $(v, v) \in G$, then the singleton $\{v\}$ is a loop.

A *strongly connected component* of $G = (V, E)$ is a maximal subgraph $G' = (V', E')$ of G such that V' is strongly connected. By $\mathcal{SCC}_G$ we denote the set $\{V' \mid G' = (V', E')$ is a strongly connected component of $G\}$. Note that in general some vertices may not occur in any loop of G, and will therefore not be represented in $\mathcal{SCC}_G$. We define *the SCC-partition* of G, denoted $\mathcal{SCC}_G^+$, as the set $\mathcal{SCC}_G \cup \{\{v\} \mid v \in G$ and v does not occur in a loop of $G\}$. Observe that $\mathcal{SCC}_G^+$ indeed forms a partition of V.

We define a relation $\preceq_G$ on the SCC-partition of G: for $S_1, S_2 \in \mathcal{SCC}_G^+$, $S_1 \preceq_G S_2$ if for some $v_1 \in S_1$, $v_2 \in S_2$, there is a path in G from v_2 to v_1, or if $S_1 = S_2 = \{v\}$, where v does not occur in a loop of G. Observe that for $S_1, S_2 \in \mathcal{SCC}_G$, $S_1 \preceq_G S_2$ implies that for any $v_1 \in S_1, v_2 \in S_2$, there is a path in G from v_2 to v_1. Also, $S_1 \preceq_G S_2$ and $S_2 \preceq_G S_1$ implies that $S_1 = S_2$. The order $\preceq_G$ is well-founded if G is finite. Hence $\preceq_G$ is a partial order. Note that $\preceq_G$ is now also defined on the strongly connected components of G, since $\mathcal{SCC}_G \subseteq \mathcal{SCC}_G^+$. We will often refer to a *minimal* strongly connected component of G: this is a component $G' = (V', E')$ such that there is no $V'' \in \mathcal{SCC}_G$ with $V'' \neq V'$ and $V'' \preceq_G V'$. Note that in general there may be several such minimal components.

We will sometimes encounter finite graphs G with the property that for every $v \in G$, there is a $v' \in G$ such that $(v, v') \in G$. It follows that such a graph contains

at least one loop, and therefore also at least one minimal strongly connected component. Furthermore, such a minimal strongly connected component is also minimal in $\mathcal{SCC}_G^+$, i.e., it has no outgoing edges.

When considering subgraphs of a given graph, the following simple property of their strongly connected components is often useful:

Proposition 6.2.4 *Let G be a graph, and G' a subgraph of G. Then for any $S' \in \mathcal{SCC}_{G'}^+$, there exists an $S \in \mathcal{SCC}_G^+$ such that $S' \subseteq S$.*

6.3 A standard algorithm for the computation of fixpoint expressions

In this section, we introduce an improved algorithm for the evaluation of fixpoint expressions proposed by Browne et al. [27]. The algorithm presented here makes extensive use of monotonicity considerations to reduce the complexity of evaluation. The number of steps required by this method is roughly the square root of the number of steps required by the best previously known algorithms. We propose an instantiation of Browne et al.'s algorithm for computing the models of propositional fixpoint definitions. We also provide a complexity result for the SAT(FD) problem. The work concerning Browne et al.'s algorithm in this section is strongly based on [27].

6.3.1 Introduction

Many automated finite-state verification algorithms can be viewed as fixpoint computations over a finite lattice. Examples include: model checking procedures for logics such as CTL [37, 38] and PDL [63], methods for computing strong and weak bisimulation equivalence in CCS [127], and language containment and emptiness algorithms for ω-automata [36]. Approaches based on fixpoint logics such as the propositional μ-calculus [97] are tied even more directly to fixpoint computation. With the increasing use of binary decision diagrams (BDDs) [28] for finite-state verification [31, 42, 126], fixpoint-based algorithms have become even more important, since methods that require the manipulation of individual states do not take advantage of this representation.

Numerous fixpoint calculi have been described in the literature [45, 97, 141], and Browne et al.'s [27] ideas for evaluating fixpoint expressions work with any of them. However, for correctness, the propositional μ-calculus of Kozen [97] will be used. This logic is designed for expressing properties of transition systems, and formulas in the logic (with no free propositional variables) evaluate to sets of states.

There have been many algorithms proposed for evaluating a formula of the logic with respect to a given transition system. These mostly fall into two categories: local and global. Local procedures are designed for proving that a specific state of the transition system satisfies the given formula. Because of this, it is not always necessary to examine all the states in the transition systems. However, the worst-case complexity of these approaches is generally larger than the complexity of the global methods. Tableau-based local approaches have been developed by Cleaveland [39], Stirling and Walker [169], and Winskel [181]. Andersen [4] and Larsen [102] have developed efficient local methods for a subset of the μ-calculus. Mader [111] has also proposed improvements to the tableau-based method of Stirling and Walker that seem to increase its efficiency (though Mader does not give a complexity bound). Global procedures generally work bottom-up through the formula, evaluating each subformula based on the value of its subformulas. Iteration is used to compute the fixpoints. Because of fixpoint nesting, a naive global algorithm may require about n^q steps to evaluate a formula, where n is the number of states in the transition system and q is the depth of nesting of the fixpoints. A linear time algorithm for evaluating alternation-free fixpoint expressions is given in [8]. Emerson and Lei [62] improved on this by observing that the complexity of evaluating a formula really depends only on the number of alternations of least and greatest fixpoints. That is, successively nested fixpoints of the same type do not increase the complexity of the computation. Emerson and Lei formalized this using the notion of alternation depth, and they gave an algorithm requiring only about n^d steps, where d is the alternation depth. The Browne algorithm, as represented in this section, is also a global method. By using extensive monotonicity considerations, it can be shown that only about $n^{d/2}$ steps are required to evaluate a formula with alternation depth d. Thus, this method requires only about the square root of the time needed by the earlier algorithms.

6.3.2 The propositional μ-calculus

In the propositional μ-calculus [97], formulas are built up from:

- atomic propositions $p, p_1, p_2, \ldots$,
- atomic propositional variables $R, R_1, R_2, \ldots$,
- logical connectives $\wedge$ and $\vee$,
- modal operators $\langle a \rangle$ and $[a]$, where a is one of the program letters $a, b, a_1, a_2, \ldots$, and
- fixpoint operators μR_i and νR_i.

Negations can be applied to atomic propositions, but this is not important for the purpose. Formulas in this calculus are interpreted relative to a transition system that consists of:

- a nonempty set of states $\top$ (the size of this set is denoted by n),
- a mapping L that takes each atomic proposition to some subset of $\top$ (the states where the proposition is true),
- a mapping T that takes each program letter to a binary relation over $\top$ (the state changes that can result from executing the program).

The intuitive meaning of the formula $\langle a \rangle \phi$ is "it is possible to execute a and transition to a state where ϕ holds". $[\cdot]$ is the dual of $\langle \cdot \rangle$. For $[a]\phi$, the intended meaning is that "ϕ holds in all states reachable (in one step) by executing a". The μ and ν operators are used to express least and greatest fixpoints, respectively. To emphasize the duality between least and greatest fixpoints, we write the empty set of states as $\bot$.

Formally, a formula ϕ depending on free propositional variables $R_1, R_2, \ldots, R_k$ is interpreted as a k-argument predicate transformer. (A predicate transformer is simply a mapping from sets of states to a set of states.) We write this predicate transformer as ϕ^M. ϕ^M is defined inductively by giving its value $\phi^M(\bar{S})$ for a vector $\bar{S} = (S_1, \ldots, S_k)$ of arguments.

1. $p^M(\bar{S}) = L(p)$.
2. $R_i^M(\bar{S}) = S_i$.
3. $(\phi \wedge \psi)^M(\bar{S}) = \phi^M(\bar{S}) \cap \psi^M(\bar{S})$. Disjunction is similar.
4. $(\langle a \rangle \phi)^M(\bar{S}) = \{s \mid \exists t((s,t) \in T(a) \wedge t \in \phi^M(\bar{S}))\}$.
   $([a]\phi)^M(\bar{S}) = \{s \mid \forall t((s,t) \in T(a) \supset t \in \phi^M(\bar{S}))\}$.
5. $(\mu R.\phi)^M(\bar{S})$ is defined to be least fixpoint of the predicate transformer $\tau : 2^\top \to 2^\top$ defined by
   $$\tau(S) = \phi^M(S, \bar{S}),$$
   where the first parameter of ϕ^M is the value for R. The interpretation of $\nu R.\phi$ is similar, except that we take the greatest fixpoint.

Within formulas, there is no negation (except potentially on the atomic propositions), and so the fixpoints are guaranteed to be well-defined. Formally, each possible τ is monotonic ($S \subseteq S'$ implies $\tau(S) \subseteq \tau(S')$). This is enough to ensure the existence of the fixpoints [174]. Further, since we will be evaluating formulas

only over finite transition systems, monotonicity of τ implies that τ is also $\cup$-continuous and $\cap$-continuous, and hence the least and greatest fixpoints can be computed by iterative evaluation:

$$(\mu R.\phi)^M(\bar{S}) = \bigcup_i \tau^i(\bot), \qquad (\nu R.\phi)^M(\bar{S}) = \bigcap_i \tau^i(\top).$$

Since the domain is finite, the iteration must stop after a finite number of steps. More precisely, for some $i \leq |\top|$, the fixpoint is equal to $\tau^i(\bot)$ (for a least fixpoint) or $\tau^i(\top)$ (for a greatest fixpoint). To find the fixpoint, we repeatedly apply τ starting from $\bot$ or from $\top$ until the result does not change.

Since we will be using the concept of alternation depth, we briefly summarize Emerson and Lei's observations [62]. Consider the expression

$$\mu R_1.(\langle a \rangle R_1) \vee (\mu R_2.R_1 \vee p \vee \langle b \rangle R_2).$$

The subformula $\mu R_2.(\cdots)$ defines a monotonic predicate transformer τ taking one set (the value of R_1) to another (the value of the least fixpoint of R_2). When evaluating the outer fixpoint, we start with the approximation $\bot$ and then compute $\tau(\bot)$. Now R_1 is increased (say to S_1), and we want to compute the least fixpoint $\tau(S_1)$. Since $\bot \subseteq S_1$, by monotonicity we know that $\tau(\bot) \subseteq \tau(S_2)$. To compute a least fixpoint, it is enough to start iterating with any approximation known to be below the fixpoint. Thus here, we can start iterating with $\tau(\bot)$ instead of $\bot$. At the next step, R_1 will be even larger, and so we will start the inner fixpoint computation with $\tau(S_1)$. We never restart the inner fixpoint computation, and so we can have at most about n increases in the value of the inner fixpoint variable. Overall, we only need about n steps to evaluate this expression, instead of n^2.

Emerson and Lei show that this type of simplification makes it possible to evaluate a formula ϕ in about n^d steps, where d is the alternation depth of the formula. The alternation depth of a formula is intuitively equal to the number of alternating nestings of least and greatest fixpoints. Formally, the alternation depth is defined as follows. Assume for simplicity that the formula does not contain any nonatomic subformulas which do not contain free propositional variables (these can be independently evaluated and then treated as atomic propositions).

- The alternation depth of an atomic proposition or propositional variable is 0.
- The alternation depth for formulas like $\phi \wedge \psi$, $\phi \vee \psi$, $\langle a \rangle \phi$ etc., is the maximum alternation depth of the subformulas ϕ, ψ, etc.
- The alternation depth of $\mu R.\phi$ is the maximum of: one, the alternation depth of ϕ, and one plus the alternation depth of any top-level ν-subformulas of

ϕ. A top-level ν-subformula of ϕ is a subformula $\nu R'.\psi$ of ϕ that is not contained within any other fixpoint subformula of ϕ. The alternation depth for $\nu R.\phi$ is similarly defined.

6.3.3 The algorithm

To simplify notation, we consider a fixpoint computation of the form:

$$\begin{aligned}
&F_1 \equiv \mu R_1.\psi_1(R_1, F_2),\\
&F_2 \equiv \nu R_2.\psi_2(R_1, R_2, F_3),\\
&F_3 \equiv \mu R_3.\psi_3(R_1, R_2, R_3, F_4),\\
&F_4 \equiv \nu R_4.\psi_4(R_1, R_2, R_3, R_4, F_5),\\
&\vdots\\
&F_d \equiv \sigma_d R_d.\psi_d(R_1, R_2, \ldots, R_d),
\end{aligned}$$

where $\equiv$ denotes syntactic equality. Note that d is the alternation depth of this formula. We write $\sigma_d R_k.(\cdots)$ to mean $\mu R_k.(\cdots)$ if R_k is given by a least fixpoint, and to mean $\nu R_k.(\cdots)$ otherwise. Define $\phi_d \equiv \psi_d$, and let

$$\phi_k(R_1, \ldots, R_k) \equiv \psi_k(R_1, \ldots, R_k, F_{k+1})$$

for $k < d$. Note that ψ_k is a formula with $k+1$ free propositional variables (except for ψ_d); the last parameter gives the value of the inner fixpoint. Then ϕ_k is obtained from ψ_k by instantiating the last parameter with the inner fixpoint. For notational simplicity, we will identify syntactic formulas with their interpretations in the discussion below. So, depending on context, a formula like ϕ_k may mean either the actual formula or the k-argument predicate transformer ϕ_k^M.

The basic idea

Before going into details of Browne et al.'s algorithm [27], we illustrate the idea on a formula involving three fixpoints:

$$\mu R_1.\psi_1(R_1, \nu R_2.\psi_2(R_1, R_2, \mu R_3.\psi_3(R_1, R_2, R_3))).$$

To compute the outer fixpoint, we start with $R_1 = \bot, R_2 = \top$ and $R_3 = \bot$. Call these values R_1^0, R_2^{00}, and R_3^{000}, respectively. The superscript on R_k gives the iteration indices for the fixpoints involving $R_1, \ldots, R_k$. We then iterate to compute the inner fixpoint; call the value of this fixpoint $R_3^{00\omega}$. We now compute the next approximation R_2^{01} for R_2 by evaluating $\psi_2(R_1^0, R_2^{0\omega}, R_3^{0\omega\omega})$ and go back to the inner fixpoint. Eventually, we reach the fixpoint for R_2, having computed $R_2^{00}, R_3^{00\omega}, R_2^{01}, R_3^{01\omega}, \ldots, R_2^{0\omega}, R_3^{0\omega\omega}$. Now we proceed to $R_1^1 = \psi_1(R_1^0, R_2^{0\omega}, R_3^{0\omega\omega})$.

We know that $R_1^0 \subseteq R_1^1$, and we are now going to compute $R_2^{1\omega}$. Note that the values $R_2^{0\omega}$ and $R_2^{1\omega}$ are given by

$$R_2^{0\omega} = \nu R_2.\psi_2(R_1^0, R_2, \mu R_3.\psi_3(R_1^0, R_2, R_3)),$$

and

$$R_2^{1\omega} = \nu R_2.\psi_2(R_1^1, R_2, \mu R_3.\psi_3(R_1^1, R_2, R_3)).$$

By monotonicity, we know that $R_2^{1\omega}$ will be a superset of $R_2^{0\omega}$. However, since R_2 is computed by a greatest fixpoint, this information does not help; we still must start computing with $R_2^{10} = \top$. At this point, we begin to compute the inner fixpoint again. But now let us look at $R_3^{00\omega}$ and $R_3^{10\omega}$. We have

$$R_3^{00\omega} = \mu R_3.\psi_3(R_1^0, R_2^{00}, R_3),$$
$$R_3^{10\omega} = \mu R_3.\psi_3(R_1^1, R_2^{10}, R_3).$$

Since $R_1^0 \subseteq R_1^1$ and $R_2^{00} \subseteq R_2^{10}$, monotonicity implies that $R_3^{00\omega} \subseteq R_3^{10\omega}$. Now R_3 is a least fixpoint, so starting the computation of $R_3^{10\omega}$ anywhere below the fixpoint value is acceptable. Thus, we can start the computation for $R_3^{10\omega}$ with $R_3^{10\omega} = R_3^{00\omega}$. Since $R_3^{00\omega}$ is in general larger than $\perp$, we obtain faster convergence. Also note that since $R_1^0 \subseteq R_1^1$ and $R_3^{00\omega} \subseteq R_3^{10\omega}$ we will have $R_2^{01} \subseteq R_2^{11}$. This means that we can use the same trick when computing $R_3^{11\omega}$: we start the computation from $R_3^{110} = R_3^{01\omega}$. In general, we can start computing $R_3^{1j\omega}$ from $R_3^{1j0} = R_3^{0j\omega}$. Similarly, once we find R_1^2 (or in general, R_1^{k+1}), we can start computing the inner fixpoints from $R_3^{1m\omega}$ ($R_3^{km\omega}$).

If we use this idea, how many steps does the computation take? The dominating term is the number of steps made when computing the inner fixpoint. With previously known algorithms, this inner computation starts from $\perp$ each time, and hence may involve about n^3 steps (one factor of n for each of the three fixpoints). In our case, if we fix a particular j, then we have

$$R_3^{0j0} \subseteq R_3^{0j\omega} = R_3^{1j0} \subseteq R_3^{1j\omega} = R_3^{2j0} \subseteq \ldots = R_3^{\omega j0} \subseteq R_3^{\omega j\omega}.$$

This implies that for each j, we can have at most n strict inclusions among the values of R_3^{ijm} that we compute, and so for each j we take only about n steps. Since there can be up to n different j values, we take only about n^2 steps while computing the inner fixpoint, thus saving a factor of n.

The relationship between the different approximations to R_3 is shown in Figure 6.1. The computation of least fixpoints proceeds from bottom to top, and the computation of greatest fixpoints proceeds from left to right. When computing with approximation R_1^j, we save the "frontier" values $R_3^{j-\omega}$ and use them as the initial approximations $R_3^{(j+1)-0}$ when computing with R_1^{j+1}. We have at most n strict inclusions within each vertical chain in the figure.

$$
\begin{array}{ccccccc}
R_3^{\omega 0\omega} & \supseteq & R_3^{\omega 1\omega} & \supseteq & \ldots & \supseteq & R_3^{\omega\omega\omega} \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
\vdots & & \vdots & & & & \vdots \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
R_3^{\omega 01} & \supseteq & R_3^{\omega 11} & \supseteq & \ldots & \supseteq & R_3^{\omega\omega 1} \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
R_3^{\omega 00} & \supseteq & R_3^{\omega 10} & \supseteq & \ldots & \supseteq & R_3^{\omega\omega 0} \\
\|\!\cup & & \|\!\cup & & & & \|\!\cup \\
\vdots & & \vdots & & & & \vdots \\
\|\!\cup & & \|\!\cup & & & & \|\!\cup \\
R_3^{10\omega} & \supseteq & R_3^{11\omega} & \supseteq & \ldots & \supseteq & R_3^{1\omega\omega} \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
\vdots & & \vdots & & & & \vdots \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
R_3^{101} & \supseteq & R_3^{111} & \supseteq & \ldots & \supseteq & R_3^{1\omega 1} \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
R_3^{100} & \supseteq & R_3^{110} & \supseteq & \ldots & \supseteq & R_3^{1\omega 0} \\
\|\!\cup & & \|\!\cup & & & & \|\!\cup \\
R_3^{00\omega} & \supseteq & R_3^{01\omega} & \supseteq & \ldots & \supseteq & R_3^{0\omega\omega} \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
\vdots & & \vdots & & & & \vdots \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
R_3^{001} & \supseteq & R_3^{011} & \supseteq & \ldots & \supseteq & R_3^{0\omega 1} \\
\cup\!| & & \cup\!| & & & & \cup\!| \\
R_3^{000} & \supseteq & R_3^{010} & \supseteq & \ldots & \supseteq & R_3^{0\omega 0}
\end{array}
$$

Figure 6.1: Relationships between approximations for R_3

Note that we can build this type of table for arbitrarily nested fixpoints. Suppose, for example, that we were also computing an outer greatest fixpoint for a relation R_0. Figure 6.1 would correspond to a series of computations with R_0 at $\top$. If we then compute the next approximation for R_0, it will be smaller than the initial approximation. Then by monotonicity, when we go through the computations for R_1, R_2, and R_3 again, we will get at each stage something smaller than during the first set of computations. For R_2, this means that we can use the frontier fixpoint values produced during the first set of computations as initial approximations when doing the second set of computations. The effect is to build a second table like the one in the figure to the right of the previous table. This process would be repeated for each new approximation for R_0. As before, we could argue that the number of strict inclusions along any chain (now

running horizontally) would be bounded by n. At first it seems that these ideas must lead to a polynomial time algorithm for evaluating formulas of arbitrary alternation depth. This appears reasonable because we have a bound of n on the number of strict inclusions going both horizontally and vertically, so the number of distinct entries in a table should be about n^2. Unfortunately, this intuition is not correct. The problem arises because the chains may not "line up" due to fixpoint computations converging in less than n steps. The result is that we can only guarantee that the algorithm will take no more than about $n^{d/2}$ steps.

A simple version of the algorithm

Now we turn to the details of Browne et al.'s algorithm. For reasons of notational simplicity, the first approach that we will describe will be one that always iterates n times when computing a fixpoint, even when convergence is achieved earlier. It will also only save frontier values for least fixpoints. Afterwards we will prove that terminating the computation of least fixpoints once convergence is achieved is allowable. This will be enough to give us the desired time bound on the algorithm. We will then give the general algorithm that saves frontier values and terminates once convergence is achieved for both types of fixpoints.

As in the discussion earlier, we will superscript relation names with vectors of iteration indices to show various approximations. Each iteration index will be a number between 0 and n, inclusive. We will let $\bar{\jmath}$ and $\bar{\imath}$ denote vectors of iteration indices. For example, we could write R_3^{000} as $R_3^{\bar{\jmath}}$ with $\bar{\jmath} = 000$. When we write such an expression, it is implicit that the length of the vector corresponds in the right way to the depth of nesting of the fixpoint. Variables j and i will denote individual iteration indices. By the juxtaposition a vector of iteration indices with some expressions denoting individual indices, we mean the vector formed by concatenating the values of the expressions onto the end of the vector. So if $\bar{\jmath} = 00$, then $R_3^{\bar{\jmath}0}$ would mean R_3^{000}. The notation $\bar{0}$ will indicate a vector of all zeros of an appropriate length. Given a vector $\bar{\jmath}$ of iteration indices, the notation $\mu(\bar{\jmath})$ will be the vector of indices that correspond to least fixpoint variables. In our earlier example, R_1 and R_3 are the least fixpoints, so $\mu(123)$ would denote the vector 13. Similarly, $\nu(\bar{\jmath})$ selects the indices corresponding to greatest fixpoints. We will also need a partial order on vectors of indices. The notation $\bar{\imath} \preceq \bar{\jmath}$ will mean that $\bar{\imath}$ and $\bar{\jmath}$ have the same length, that $\nu(\bar{\imath}) = \nu(\bar{\jmath})$, and that $\mu(\bar{\imath})$ is lexicographically less than or equal to $\mu(\bar{\jmath})$. If the first and third indices represent least fixpoints, then $312 \preceq 410$. The relation $\preceq$ holds between successive elements in the vertical chains in Figure 6.1. The notation $p(\bar{\jmath})$ will denote the immediate predecessor of $\bar{\jmath}$ under $\preceq$. This exists whenever $\mu(\bar{\jmath}) \neq \bar{0}$. Note that $\nu(p(\bar{\jmath})) = \nu(\bar{\jmath})$, and that $\mu(p(\bar{\jmath}))$ is the immediate predecessor of $\mu(\bar{\jmath})$ in the lexicographic ordering. The notation $h(\bar{\jmath}, l)$ will denote the vector of length l that agrees with $\bar{\jmath}$ on the initial indices. As an example, $h(312, 2) = 31$.

The algorithm will consist of computing the following approximations:

1. If $\sigma_k = \nu$:
   (a) $R_k^{\bar{\jmath}0} = \top$.
   (b) $R_k^{\bar{\jmath}(j+1)} = \psi_k(R_1^{h(\bar{\jmath}j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}j,k-1)}, R_k^{\bar{\jmath}j}, R_{k+1}^{\bar{\jmath}jn})$. (Note that $R_{k+1}^{\bar{\jmath}jn}$ is an (inner) fixpoint since we must have convergence in at most n steps.)
2. If $\sigma_k = \mu$:
   (a) $R_k^{\bar{\jmath}0} = \bot$ if $\mu(\bar{\jmath}0) = \bar{0}$.
   (b) $R_k^{\bar{\jmath}0} = R_k^{p(\bar{\jmath}0)}$ if $\mu(\bar{\jmath}0) = \bar{0}$.
   (c) $R_k^{\bar{\jmath}(j+1)} = \psi_k(R_1^{h(\bar{\jmath}j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}j,k-1)}, R_k^{\bar{\jmath}j}, R_{k+1}^{\bar{\jmath}jn})$.

The evaluation is to be done in the computation order. *Computation order* is simply the natural order of evaluation. Formally, it is the total order on vectors of iteration indices of potentially different lengths defined as follows. $\bar{\imath}$ occurs before $\bar{\jmath}$ in computation order when:

- $\bar{\imath}$ and $\bar{\jmath}$ have the same length and $\bar{\imath}$ strictly precedes $\bar{\jmath}$ lexicographically, or
- $\bar{\imath}$ is shorter than $\bar{\jmath}$, and $\bar{\imath}$ is lexicographically less than or equal to $h(\bar{\jmath}, |\bar{\imath}|)$, or
- $\bar{\jmath}$ is shorter than $\bar{\imath}$, and $h(\bar{\imath}, |\bar{\jmath}|)$ strictly precedes $\bar{\jmath}$ lexicographically.

For our earlier three relation example, computation order is: $0, 00, 000, 001, \ldots, 00n$, $01, 010, \ldots, 0nn, 1, 10$, etc. Thus, we would compute $R_1^0, R_2^{00}, R_3^{000}, \ldots, R_3^{00n}, R_2^{01}$, etc. The result returned from the algorithm is R_1^n. Notice that $\bar{\imath} \preceq \bar{\jmath}$ implies that $\bar{\imath}$ preceded $\bar{\jmath}$ in the computation order.

Proof of correctness

The proof of the correctness of the above algorithm, as shown below, is completely copied from [27]. As a preliminary step, we show that the elements in a vertical chain are related by set inclusion.

Lemma 6.3.1 *For all $\bar{\imath}, \bar{\jmath}$ and k, if $\bar{\imath} \preceq \bar{\jmath}$ then $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$.*

Proof. The proof proceeds by induction on $\bar{\jmath}$ in the computation order.

1. If $\sigma_k = \nu$ and $\bar{\jmath} = \bar{\jmath}'0$, then $R_k^{\bar{\jmath}} = \top$, and hence $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$ for all $\bar{\imath} \preceq \bar{\jmath}$.

2. Suppose $\sigma_k = \nu$ and $\bar{\jmath} = \bar{\jmath}'(j+1)$. Then

$$R_k^{\bar{\jmath}} = \psi_k(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'j,k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$

Since $\bar{\imath} \preceq \bar{\jmath}$, we know that $\bar{\jmath}$ and $\bar{\imath}$ have the same values for the ν indices. This implies that $\bar{\imath} = \bar{\imath}'(j+1)$ and that $R_k^{\bar{\imath}}$ must be defined by

$$R_k^{\bar{\imath}} = \psi_k(R_1^{h(\bar{\imath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\imath}'j,k-1)}, R_k^{\bar{\imath}'j}, R_{k+1}^{\bar{\imath}'jn}).$$

Now $h(\bar{\imath}'j, l) \subseteq h(\bar{\jmath}'j, l)$ for $l < k$, and so by the induction hypothesis

$$R_l^{h(\bar{\imath}'j,l)} \subseteq R_l^{h(\bar{\jmath}'j,l)}.$$

Similarly, $R_k^{\bar{\imath}'j} \subseteq R_k^{\bar{\jmath}'j}$ and $R_{k+1}^{\bar{\imath}'jn} \subseteq R_{k+1}^{\bar{\jmath}'jn}$. Since ψ_k is monotonic, $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$.

3. If $\sigma_k = \mu$ and $\mu(\bar{\jmath}) = \bar{0}$, then $R_k^{\bar{\jmath}} = \bot$. Since $\bar{\imath} \preceq \bar{\jmath}, \mu(\bar{\imath}) = \bar{0}$, so $R_k^{\bar{\imath}} = \bot$ also.

4. Suppose $\sigma_k = \mu, \mu(\bar{\jmath}) \neq \bar{0}$, and $\bar{\jmath}$ has the form $\bar{\jmath}'0$. If $\bar{\imath} = \bar{\jmath}$, then $R_k^{\bar{\imath}} = R_k^{\bar{\jmath}}$. Otherwise, $\bar{\imath} \preceq p(\bar{\jmath})$. Then by the induction hypothesis, $R_k^{\bar{\imath}} \subseteq R_k^{p(\bar{\jmath})}$. But by definition $R_k^{\bar{\jmath}} = R_k^{p(\bar{\jmath})}$, and so $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$.

5. If $\sigma_k = \mu$ and $\bar{\jmath}$ has the form $\bar{\jmath}'(j+1)$, then

$$R_k^{\bar{\jmath}} = \psi_k(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'j,k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$

When $\bar{\imath} = \bar{\jmath}$, we have $R_k^{\bar{\imath}} = R_k^{\bar{\jmath}}$; otherwise, $\bar{\imath} \preceq \bar{\jmath}'j$. By the induction hypothesis, $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}'j}$. Hence, it suffices to show that $R_k^{\bar{\jmath}'j} \subseteq R_k^{\bar{\jmath}'(j+1)}$. If $j > 0$, then

$$R_k^{\bar{\jmath}'j} = \psi_k(R_1^{h(\bar{\jmath}'(j-1),1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'(j-1),k-1)}, R_k^{\bar{\jmath}'(j-1)}, R_{k+1}^{\bar{\jmath}'(j-1)n}).$$

Now $h(\bar{\jmath}'(j-l), l) = h(\bar{\jmath}'j, l)$ for $l < k$, and hence $R_l^{h(\bar{\jmath}'(j-l),l)} = R_l^{h(\bar{\jmath}'j,l)}$. Since $\bar{\jmath}'(j-1) \preceq \bar{\jmath}'j$, the induction hypothesis implies $R_k^{\bar{\jmath}'(j-1)} \subseteq R_k^{\bar{\jmath}'j}$. Similarly, $R_{k+1}^{\bar{\jmath}'(j-1)n} \subseteq R_{k+1}^{\bar{\jmath}'jn}$. Then monotonicity of ψ_k implies that $R_k^{\bar{\jmath}'j} \subseteq R_k^{\bar{\jmath}}$ which is the desired result.

For the case $j = 0$, we want to show that $R_k^{\bar{\jmath}'0} \subseteq R_k^{\bar{\jmath}'1}$. We have two cases, depending on the value of $\mu(\bar{\jmath}'0)$.

(a) If $\mu(\bar{\jmath}'0) = \bar{0}$, then $R_k^{\bar{\jmath}'0} = \bot$, and trivially $R_k^{\bar{\jmath}'0} \subseteq R_k^{\bar{\jmath}1}$.

(b) If $\mu(\bar{\jmath}'0) \neq \bar{0}$, then $R_k^{\bar{\jmath}'0} = R_k^{p(\bar{\jmath}'0)}$. Define $\bar{\jmath}''$ so that $\bar{\jmath}''n = p(\bar{\jmath}'0)$. By definition,

$$R_k^{\bar{\jmath}'0} = \psi_k(R_1^{h(\bar{\jmath}''(n-1),1)}, \ldots, R_{k-1}^{h(\bar{\jmath}''(n-1),k-1)}, R_k^{\bar{\jmath}''(n-1)}, R_{k+1}^{\bar{\jmath}''(n-1)n}).$$

Also
$$R_k^{\bar{\jmath}'1} = \psi_k(R_1^{h(\bar{\jmath}'0,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'0,k-1)}, R_k^{\bar{\jmath}'0}, R_{k+1}^{\bar{\jmath}'0n}).$$
Now $h(\bar{\jmath}''(n-l), l) \preceq h(\bar{\jmath}'0, l)$ for $l < k$, and so the induction hypothesis implies $R_l^{h(\bar{\jmath}''(n-l),l)} \subseteq R_l^{h(\bar{\jmath}'0,l)}$. Similarly, $R_k^{\bar{\jmath}''(n-1)} \subseteq R_k^{\bar{\jmath}'0}$ and $R_{k+1}^{\bar{\jmath}''(n-1)n} \subseteq R_{k+1}^{\bar{\jmath}'0n}$. Then by monotonicity of ψ_k, we have $R_k^{\bar{\jmath}'0} \subseteq R_k^{\bar{\jmath}'1}$, which is the desired result.

Now we show that the fixpoints are correctly computed.

Theorem 6.3.2 *For all $\bar{\jmath}$ and k, if $\bar{\jmath}$ has the form $\bar{\jmath}'n$, then*
$$R_k^{\bar{\jmath}} = \sigma_k R_k.\phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$
In particular,
$$R_1^n = \mu R_1.\phi_1(R_1).$$

Proof. Again, we proceed by induction on $\bar{\jmath}$ in computation order.

1. Assume $\sigma_k = \nu$. $R_k^{\bar{\jmath}}$ is computed by the sequence $R_k^{\bar{\jmath}'0} = \top, R_k^{\bar{\jmath}'1}, \ldots, R_k^{\bar{\jmath}'n}$. We have
$$R_k^{\bar{\jmath}'(j+1)} = \psi_k(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'j,k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$
Note that $h(\bar{\jmath}'j, l) = h(\bar{\jmath}', l)$ for $l < k$. Hence we can rewrite the above equation as
$$R_k^{\bar{\jmath}'(j+1)} = \psi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$
By the induction hypothesis,
$$R_{k+1}^{\bar{\jmath}'jn} = \sigma_{k+1} R_{k+1}.\phi_{k+1}(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_k^{h(\bar{\jmath}'j,k)}, R_{k+1}).$$
Now $h(\bar{\jmath}'j, l) = h(\bar{\jmath}', l)$ for $l < k$, and we also have that $h(\bar{\jmath}'j, k) = \bar{\jmath}'j$. Substituting gives
$$R_{k+1}^{\bar{\jmath}'jn} = \sigma_{k+1} R_{k+1}.\phi_{k+1}(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}).$$
Now we see that $R_k^{\bar{\jmath}'0}, \ldots, R_k^{\bar{\jmath}'n}$ is just the standard iterative computation of the fixpoint. Since we must have convergence within n steps,
$$R_k^{\bar{\jmath}'n} = \nu R_k.\phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

2. The case when $\sigma = \mu$ and $\mu(\bar{\jmath}') = \bar{0}$ is analogous to the previous one: we start with $R_k^{\bar{\jmath}'0} = \bot$ and compute the standard series of approximations.

3. Assume $\sigma = \mu$ and $\mu(\bar{\jmath}') \neq \bar{0}$. Here, we start the computation with $R_k^{\bar{\jmath}'0} = R_k^{p(\bar{\jmath}'0)}$. Define $\bar{\jmath}''$ so that $\bar{\jmath}''n = p(\bar{\jmath}'0)$. To ensure that $R_k^{\bar{\jmath}}$ is equal to the desired fixpoint, we first show that the initial approximation $R_k^{p(\bar{\jmath}'0)}$ is a subset of the fixpoint. By the induction hypothesis, we know that

$$R_k^{\bar{\jmath}''n} = \mu R_k.\phi_k(R_1^{h(\bar{\jmath}'',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'',k-1)}, R_k).$$

We want to know that this is a subset of

$$\mu R_k.\phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

By the definition of $\bar{\jmath}''$, we have that $\bar{\jmath}'' \preceq \bar{\jmath}'$. Hence $h(\bar{\jmath}'', l) \preceq h(\bar{\jmath}', l)$ for all $l < k$. Using the previous lemma, we see that $R_l^{h(\bar{\jmath}'',l)} \subseteq R_l^{h(\bar{\jmath}',l)}$. Then the monotonicity implies

$$R_k^{\bar{\jmath}''n} \subseteq \mu R_k.\phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

Now we simply proceed as in the previous two cases, using the fact that to compute a least fixpoint with iteration, it is sufficient that the initial approximation is a subset of the fixpoint.

As mentioned, the algorithm given above does not stop the computation of a fixpoint when convergence is achieved. As a result, it will take about n^d steps when evaluating the alternating fixpoint formula that we are considering. To see what potential problem might arise with terminating the computation early, let us consider how an implementation of the algorithm would work. The idea will be to keep a table of frontier values for each least fixpoint. This table is indexed by the iteration indices for the enclosing greatest fixpoints, and initially all the entries are $\bot$. When computing a least fixpoint, we take the table value corresponding to the current greatest fixpoint iteration indices and use that as our initial approximation. After computing the fixpoint, we store the result back in the same spot in the table. In our earlier example with three fixpoints, the table for R_3 would have $n+1$ entries, since there is one enclosing greatest fixpoint. During the computation of the R_2^{0-}, the table entries will be used and replaced. Just after we compute R_1^1, the jth entry in the table will contain the fixpoint value R_3^{0jn}. Suppose now that we cut off the computation of R_2^{0-} as soon as convergence is detected. Then some of the table values for R_3 will not be updated. While it is not difficult in the case of our example with three fixpoint expressions, in general it is hard to argue that "stale" table values will not cause problems. Fortunately, we can easily prove that when computing a least fixpoint, we can stop as soon as convergence is detected. To

do this, we just need to show that once we obtain convergence, none of the table values for the inner fixpoints can change. More precisely, we consider a situation in which $R_k^{\bar{\jmath}(j+1)} = R_k^{\bar{\jmath}j}$ for the least fixpoint variable R_k. A table value for an inner least fixpoint variable R_l after the computation of $R_k^{\bar{\jmath}l}$ is of the form $R_l^{\bar{\jmath}'}$, where $\bar{\jmath}'$ agrees with $\bar{\jmath}j$ on the first k indices and where $\mu(\bar{\jmath}') = \mu(\bar{\jmath}j)n \ldots n$ (the indices for least fixpoint variables nested between R_k and R_l are n). The same table value after the computation of $R_k^{\bar{\jmath}(j+1)}$ is $R_l^{\bar{\jmath}''}$, where $\bar{\jmath}''$ differs from $\bar{\jmath}'$ only at position k, and at that position, $\bar{\jmath}''$ has $j+1$ instead of j. We claim that these two table values must be equal.

Theorem 6.3.3 *Assume $\sigma_k = \mu$ and $R_k^{\bar{\jmath}(j+1)} = R_k^{\bar{\jmath}j}$. Then for all $l > k$ and $\bar{\jmath}'$ and $\bar{\jmath}''$ such that:*

1. $h(\bar{\jmath}', k-1) = h(\bar{\jmath}'', k-1) = \bar{\jmath}$,
2. $\mu(\bar{\jmath}') = \mu(\bar{\jmath}j)n \ldots n$,
3. $\mu(\bar{\jmath}'') = \mu(\bar{\jmath}(j+1))n \ldots n$, *and*
4. $\nu(\bar{\jmath}') = \nu(\bar{\jmath}'')$,

we have $R_l^{\bar{\jmath}'} = R_l^{\bar{\jmath}''}$. (Note that we are not requiring $\sigma_l = \nu$ here.)

Proof. We proceed by induction on $\bar{\jmath}'$ in computation order. Note that since $\bar{\jmath}'$ and $\bar{\jmath}''$ differ only on the index for R_k, fixing $\bar{\jmath}'$ also fixes $\bar{\jmath}''$.

1. Assume $\sigma_l = \nu$ and $\bar{\jmath}'$ has the form $\bar{\imath}'0$. Then $\bar{\jmath}''$ has the form $\bar{\imath}''0$, and so $R_l^{\bar{\jmath}'} = R_l^{\bar{\jmath}''} = \top$.
2. Assume $\sigma_l = \nu$ and $\bar{\jmath}'$ has the form $\bar{\imath}'(m+1)$. Let $\bar{\jmath}''$ have the form $\bar{\imath}''(m+1)$. We have
$$R_l^{\bar{\jmath}'} = \psi_l(R_1^{h(\bar{\imath}'m,1)}, \ldots, R_{l-1}^{h(\bar{\imath}'m,l-1)}, R_l^{\bar{\imath}'m}, R_{l+1}^{\bar{\imath}'mn})$$
and
$$R_l^{\bar{\jmath}''} = \psi_l(R_1^{h(\bar{\imath}''m,1)}, \ldots, R_{l-1}^{h(\bar{\imath}''m,l-1)}, R_l^{\bar{\imath}''m}, R_{l+1}^{\bar{\imath}''mn}).$$
Now $h(\bar{\imath}'m, o) = h(\bar{\imath}''m, o)$ for $o < k$. By hypothesis, $R_k^{h(\bar{\imath}'m,k)} = R_k^{h(\bar{\imath}''m,k)}$ (we have that $h(\bar{\imath}'m, k) = \bar{\jmath}j$ and $h(\bar{\imath}''m, k) = \bar{\jmath}(j+1)$). By the induction hypothesis, $R_o^{h(\bar{\imath}'m,o)} = R_o^{h(\bar{\imath}''m,o)}$ for $k < o < l$. Also by the induction hypothesis, $R_l^{\bar{\imath}'m} = R_l^{\bar{\imath}''m}$ and $R_{l+1}^{\bar{\imath}'mn} = R_{l+1}^{\bar{\imath}''mn}$. Hence $R_l^{\bar{\jmath}'} = R_l^{\bar{\jmath}''}$.
3. Assume $\sigma_l = \mu$. Let $\bar{\imath}'$ and $\bar{\imath}''$ be chosen so that $\bar{\jmath}' = \bar{\imath}'n$ and $\bar{\jmath}'' = \bar{\imath}''n$. By the previous theorem, we have
$$R_l^{\bar{\jmath}'} = \mu R_l.\phi_l(R_1^{h(\bar{\imath}',1)}, \ldots, R_{l-1}^{h(\bar{\imath}',l-1)}, R_l)$$

and

$$R_l^{\bar{\jmath}''} = \mu R_l.\phi_l(R_1^{h(\bar{\imath}'',1)}, \ldots, R_{l-1}^{h(\bar{\imath}'',l-1)}, R_l).$$

As in the previous case, we have: $h(\bar{\imath}', o) = h(\bar{\imath}'', o)$ for $o < k$; $R_k^{h(\bar{\imath}',k)} = R_k^{h(\bar{\imath}'',k)}$; and $R_o^{h(\bar{\imath}',o)} = R_o^{h(\bar{\imath}'',o)}$ for $k < o < l$. Hence $R_l^{\bar{\jmath}'} = R_l^{\bar{\jmath}''}$.

Note that this result, combined with our earlier correctness theorem and monotonicity lemma, implies that once $R_k^{\bar{\jmath}(j+1)} = R_k^{\bar{\imath}j}$ then $R_l^{\bar{\imath}} = R_l^{\bar{\imath}'}$ for any $\bar{\imath}$ and $\bar{\imath}'$ with $\nu(\bar{\imath}) = \nu(\bar{\imath}')$ and with $\bar{\imath}$ and $\bar{\imath}'$ between $\bar{\jmath}(j+1)$ and $\bar{\jmath}n \ldots n$ in computation order. In other words, no table values will change if we keep iterating for R_k, and hence we can safely stop this part of the computation.

Complexity analysis

Now we analyze the complexity of Browne et al.'s algorithm. We will just be counting the number of approximations produced to avoid details such as how relations are represented. Let T_k denote the maximum number of approximations that can be produced for R_k. Obviously $T_1 = n+1$ and $T_2 = (n+1)^2$. If T_{k+1} is a greatest fixpoint, then each time we evaluate it, we produce $n+1$ approximations. The fixpoint can be evaluated at most T_k times, so when $\sigma_{k+1} = \nu$, $T_{k+1} \leq (n+1)T_k$. For $\sigma_{k+1} = \mu$, the fixpoint can also be evaluated at most T_k times. For this fixpoint, there are $(n+1)^{\lfloor (k+1)/2 \rfloor}$ table entries (since there are $\lfloor (k+1)/2 \rfloor$ enclosing greatest fixpoints). Over all the evaluations, each of these increases monotonically, so the total number of changes in these table values is bounded by $n(n+1)^{\lfloor (k+1)/2 \rfloor}$. During each of the T_k evaluations, we can also make one extra step to detect convergence. The overall number of steps is bounded by the sum of these numbers: $T_{k+1} \leq T_k + n(n+1)^{\lfloor (k+1)/2 \rfloor} \leq T_k + (n+1)^{1+\lfloor (k+1)/2 \rfloor}$. In our fixpoint formula, k is even for least fixpoints, so this simplifies to $T_{k+1} \leq T_k + (n+1)^{(k+2)/2}$. Overall, we get

$$T_{2k} \leq k(n+1)^{k+1}, \quad T_{2k+1} \leq (k+1)(n+1)^{k+1},$$

and so

$$T_k \leq \lceil k/2 \rceil (n+1)^{1+\lfloor k/2 \rfloor}.$$

Summing the T_k for all d of the fixpoints in our formula, we get a total of $\mathcal{O}(d^2(n+1)^{1+\lfloor d/2 \rfloor})$ steps to do the evaluation.

The general algorithm

Algorithm 1 is a general version of the algorithm that works for formulas of arbitrary form, e.g., propositional fixpoint definitions, and that saves information

Algorithm 1: The general algorithm

```
1  Function eval(φ)
2  Handle base cases, logical operations, etc. ;
3  if φ = μR.ψ(R) then
4      Set R to the top element of I_R ;
5      foreach inner ν variable R′ do
6          Push [] on F_R′ ;
7      repeat
8          foreach inner ν variable R′ do
9              Let Q be the queue on top of I_R′ ;
10             Dequeue e from Q ;
11             if Q is now empty then Enqueue e again ;
12             Push e on I_R′ ;
13         R_old := R ;
14         R := eval(ψ) ;
15         foreach inner ν variable R′ do
16             Pop e from F_R′ ;
17             Let Q be the queue on top of F_R′ ;
18             Enqueue e in Q ;
19             Pop I_R′ ;
20         if R ≠ R_old then
21             foreach inner μ variable R′ do
22                 Pop I_R′ ;
23                 Pop e from F_R′ ;
24                 Push e on I_R′ ;
25     until R = R_old;
26     Push R on F_R ;
27     return R
28 if φ = νR.ψ(R) then
29     Analogous code to the above
```

and stops iterating after detecting termination for both types of fixpoints. The potential problems with stale table values are avoided by storing frontier values in queues; the length of a queue tells us the number of approximations were produced when computing the frontier represented by the queue. The correctness of this general algorithm can be easily verified by the analogous proof of correctness of the simple version of the algorithm. In the algorithm, the frontier values for starting the fixpoint computations are stored on stacks. There are two stacks for each fixpoint variable R, one associated with the current frontier (I_R) and one associated with the frontier being constructed (F_R). Each stack element is either a set of states (representing an earlier fixpoint value), or a queue (representing a frontier). The queue elements may themselves be either queues (representing subfrontiers) or sets of states. We will write stacks using angle brackets and queues using square brackets, with the top of a stack and the head of a queue being on the left. Initially, the I_R stack for a top-level fixpoint variable R holds either $\top$ or $\bot$, depending on whether R is a greatest or least fixpoint. The stack I_R for a least fixpoint variable R nested inside k greatest fixpoints holds $\bot$ nested inside k queues ($[[\ldots[[\bot]]\ldots]]$). The initial value for a stack corresponding to a greatest fixpoint variable nested inside a number of least fixpoints is similarly defined.

6.3.4 A standard algorithm for the computation of fixpoint definitions

We now investigate an algorithm for computing a model of a propositional fixpoint definition which is based on Browne et al.'s [27] general algorithm for the evaluation of fixpoint expressions, as presented in the above section. Our algorithm performs a kind of iterated induction and the execution of the algorithm matches with the iterated induction.

To present the concrete algorithm, we introduce some concepts and notations.

Let $\mathcal{D}$ be a fixpoint definition, $a \in Def(\mathcal{D})$ and $a \leftarrow \varphi_a$ the rule defining a in $\mathcal{D}$. We use $body(a)$ to denote *the set of literals* in the body of the rule defining a. We denote by $\mathcal{D}(a)$ the subdefinition of $\mathcal{D}$ (or $\mathcal{D}$ itself) in which a is locally defined. We also define the *levels* of $\mathcal{D}$ and each subdefinition of $\mathcal{D}$. This is a characterization of the hierarchy of a fixpoint definition.

Consider a least fixpoint definition $\mathcal{D} = \lfloor\ \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n\ \rfloor$. The levels of $\mathcal{D}$ and each subdefinition (or, inner fixpoint definition) of $\mathcal{D}$ are defined by:

1. The level of $\mathcal{D}$ is 1.
2. Let $\mathcal{D}'$ be a subdefinition of $\mathcal{D}$, which is either a least fixpoint definition of the form $\lfloor\ \mathcal{R}', \Delta'_1, \ldots, \Delta'_m, \nabla'_1, \ldots, \nabla'_n\ \rfloor$ or a greatest fixpoint definition of

the form $\lceil\ \mathcal{R}', \Delta'_1, \ldots, \Delta'_m, \nabla'_1, \ldots, \nabla'_n\ \rceil$, and let the level of $\mathcal{D}'$ be n. Then the level of each Δ'_i is $n+1$ and the level of each ∇'_j is $n+1$ as well.

The level of a greatest fixpoint definition is similarly defined.

The queue $\mathcal{Q}$ used in the algorithm is a stratified First-In-First-Out (FIFO) queue. There is an FIFO queue $\mathcal{Q}_{\mathcal{D}'}$ for each subdefinition $\mathcal{D}'$ of $\mathcal{D}$ (include $\mathcal{D}$ itself) in $\mathcal{Q}$. $\mathcal{Q}$ is stratified by the levels of each subdefinition of $\mathcal{D}$ and $\mathcal{D}$ itself. Dequeuing an atom from $\mathcal{Q}$ always select an atom in a queue $\mathcal{Q}'$ which is in the deepest level of $\mathcal{Q}$. This structure of $\mathcal{Q}$ guarantees that a defined atom a is popped from $\mathcal{Q}$ to be evaluated only when all subdefinitions of $\mathcal{D}(a)$ have already reached fixpoints.

A counting mechanism is used in this algorithm. During the computation of fixpoint definitions by applying this algorithm, we have to count the number of the true literals in the body of each conjunctively defined atom, respectively, the number of the false literals in the body of each disjunctively defined atom, to evaluate the truth value of $body(a)$ for each defined atom a. Then we can check whether there is a mismatch between the truth value of a defined atom a and its body in the current interpretation. If such a mismatch occurs, we push this defined atom a to its queue $\mathcal{Q}_{\mathcal{D}(a)}$ in $\mathcal{Q}$. The truth value of a will be adjusted right after all subdefinitions of $\mathcal{D}(a)$ have reached fixpoints and the mismatch between the truth value of a and its body still persists by then.

The Algorithm 2, 3 are the concrete algorithms. We split this algorithm into two parts, namely, Algorithm 2 and Algorithm 3, due to the length of the algorithm. The procedure "ReInitializeBelow($\mathcal{D}$)" in it is used to re-initialize all the subdefinitions of $\mathcal{D}$ if there is an update in $\mathcal{D}$ after the iterated induction. The procedure "ReInitializeLFD($\mathcal{D}$)" is used to re-initialize a least fixpoint definition while the procedure "ReInitializeGFD($\mathcal{D}$)" is used to re-initialize a greatest fixpoint definition. According to Browne et al.'s general algorithm, it can be exploited that in case of reiteration due to an update in a least fixpoint definition $\mathcal{D}$, we only need to re-initialize all greatest fixpoint subdefinitions of $\mathcal{D}$, while in case of reiteration due to an update in a greatest fixpoint definition $\mathcal{D}$, we only need to re-initialize all least fixpoint subdefinitions of $\mathcal{D}$.

The following properties can easily be proven to be invariant.

1. For each defined atom $a \in \mathbf{C}_{\text{lits}}$, $CC(a) = |\{l \in body(a) \mid M(l) = \mathbf{f}\}|$.
2. For each defined atom $a \in \mathbf{D}_{\text{lits}}$, $CD(a) = |\{l \in body(a) \mid M(l) = \mathbf{t}\}|$.
3. For each defined atom $a \in \mathbf{C}_{\text{lits}}$, $M(body(a)) = \begin{cases} \mathbf{t} \text{ if } CC(a) = 0 \\ \mathbf{f} \text{ if } CC(a) \neq 0 \end{cases}$.
4. For each defined atom $a \in \mathbf{D}_{\text{lits}}$, $M(body(a)) = \begin{cases} \mathbf{t} \text{ if } CD(a) \neq 0 \\ \mathbf{f} \text{ if } CD(a) = 0 \end{cases}$.

Algorithm 2: A standard algorithm for computing fixpoint definitions I

Result: To compute the model M of a fixpoint definition $\mathcal{D}$

```
1  Let Q := ∅ ;
2  foreach subdefinition D' of D (include D itself) do
3      if D' is a GFD then
4          foreach defined atom a which is locally defined in D' do M(a) := t ;
5      else
6          foreach defined atom a which is locally defined in D' do M(a) := f ;
7  foreach defined atom a do
8      if a ∈ C_lits then
9          Let CC(a) := |{l ∈ body(a) | M(l) = f}| ;
10         if CC(a) = 0 then
11             Let Derived(a) := t ;
12         else
13             Let Derived(a) := f ;
14     else
15         Let CD(a) := |{l ∈ body(a) | M(l) = t}| ;
16         if CD(a) ≠ 0 then
17             Let Derived(a) := t ;
18         else
19             Let Derived(a) := f ;
20     if M(a) ≠ Derived(a) then Add a to Q_D(a) ;
```

5. If there is a mismatch between the truth value of a defined atom a and its rule body, then a is in the queue $\mathcal{Q}$.

Recall that the structure of $\mathcal{Q}$ guarantees that a defined atom a is dequeued to be evaluated if all subdefinitions of $\mathcal{D}(a)$ have reached fixpoints. With this property of the queue, which is used in the algorithm, and the above invariants, it can be easily shown that when we apply the algorithm on a definition $\mathcal{D}$, each step corresponds exactly to the iteration of the computation of the fixpoint model of $\mathcal{D}$. Thus, the above reasoning and the correctness of Algorithm 1 leads to the correctness of the standard algorithm for the computation of fixpoint definitions.

Proposition 6.3.4 *If the standard algorithm for the computation of fixpoint definitions, called on a fixpoint definition $\mathcal{D}$, returns M, then M is a fixpoint model of $\mathcal{D}$.*

Algorithm 3: A standard algorithm for computing fixpoint definitions II

```
21 while Q ≠ ∅ do
22     Pop an atom a from Q ;
23     if a ∈ C_lits and CC(a) = 0 then
24         Let Derived(a) := t ;
25     else if a ∈ D_lits and CD(a) ≠ 0 then
26         Let Derived(a) := t ;
27     else
28         Let Derived(a) := f ;
29     if M(a) = Derived(a) then
30         Skip ;
31     else
32         Let M(a) := Derived(a) ;
33         foreach defined atom b such that b ∈ C_lits and a ∈ body(b) do
34             if Derived(a) = t then
35                 Let CC(b) := CC(b) − 1 ;
36             else
37                 Let CC(b) := CC(b) + 1 ;
38             if M(b) ≠ t and CC(b) = 0 then
39                 Add b to Q_D(b) ;
40             else if M(b) = t and CC(b) ≠ 0 then
41                 Add b to Q_D(b) ;
42         foreach defined atom b such that b ∈ D_lits and a ∈ body(b) do
43             if Derived(a) = t then
44                 Let CD(b) := CD(b) + 1 ;
45             else
46                 Let CD(b) := CD(b) − 1 ;
47             if M(b) ≠ t and CD(b) ≠ 0 then
48                 Add b to Q_D(b) ;
49             else if M(b) = t and CD(b) = 0 then
50                 Add b to Q_D(b) ;
51         ReInitializeBelow(D(a))
52 return M
```

Procedure ReInitializeBelow($\mathcal{D}$)

```
1  if D is a greatest fixpoint definition then
2      foreach subdefinition D' of D do
3          ReInitializeGFD(D');
4  else
5      foreach subdefinition D' of D do
6          ReInitializeLFD(D');
```

Procedure ReInitializeLFD($\mathcal{D}$)

```
1  if D is a greatest fixpoint definition then
2      foreach defined atom a which is locally defined in D do
3          if M(a) ≠ t then
4              Let M(a) := t ;
5              foreach defined atom b such that b ∈ C_lits and a ∈ body(b) do
6                  Let CC(b) := CC(b) − 1 ;
7                  if M(b) ≠ t and CC(b) = 0 then
8                      Add b to Q_D(b) ;
9              foreach defined atom b such that b ∈ D_lits and a ∈ body(b) do
10                 Let CD(b) := CD(b) + 1 ;
11                 if M(b) ≠ t and CD(b) ≠ 0 then
12                     Add b to Q_D(b) ;
13 foreach subdefinition D' of D do
14     ReInitializeLFD(D')
```

Example 6.3.5 *Let* $\mathcal{D} = \left\lfloor \begin{array}{l} p \leftarrow \top \\ \lceil\ q \leftarrow p\ \rceil \end{array} \right\rfloor$.

- *Step 1:* $M(q) = \mathbf{t}$ *in Line 4 and* $M(p) = \mathbf{f}$ *in Line 6. It follows that* $CC(p) = 0$ *and* $CC(q) = 1$ *in Line 9. We also have that* $Derived(p) = \mathbf{t}$ *in Line 11 and* $Derived(q) = \mathbf{f}$ *in Line 13. Thus, add* p, q *to* $\mathcal{Q}$ *in Line 20. At the end of this step, we have that* $M = \{p \mapsto \mathbf{f}, q \mapsto \mathbf{t}\}$ *and* $\mathcal{Q} = \{p, q\}$.
- *Step 2: Pop* q *from* $\mathcal{Q}$ *in Line 22 because* q *is at a deeper level of* $\mathcal{Q}$ *than* p. $Derived(q) = \mathbf{f}$ *in Line 28 and* $M(q) = \mathbf{f}$ *in Line 32. By the end of this step, we have that* $M = \{p \mapsto \mathbf{f}, q \mapsto \mathbf{f}\}$ *and* $\mathcal{Q} = \{p\}$.

Procedure ReInitializeGFD($\mathcal{D}$)

```
1  if D is a least fixpoint definition then
2      foreach defined atom a which is locally defined in D do
3          if M(a) ≠ f then
4              Let M(a) := f ;
5              foreach defined atom b such that b ∈ C_lits and a ∈ body(b) do
6                  Let CC(b) := CC(b) + 1 ;
7                  if M(b) = t and CC(b) ≠ 0 then
8                      Add b to Q_D(b) ;
9              foreach defined atom b such that b ∈ D_lits and a ∈ body(b) do
10                 Let CD(b) := CD(b) − 1 ;
11                 if M(b) = t and CD(b) = 0 then
12                     Add b to Q_D(b) ;
13 foreach subdefinition D' of D do
14     ReInitializeGFD(D')
```

- *Step 3: Pop p from $\mathcal{Q}$ in Line 22. $Derived(p) = \mathbf{t}$ in Line 24 and then $M(p) = \mathbf{t}$ in Line 32. Because $p \in body(q)$, we have that $CC(q) = 0$ in Line 35. Since $M(q) = \mathbf{f}$ and $CC(q) = 0$, we add q to $\mathcal{Q}$ in Line 39. We then goto Line 51 to do ReInitializeBelow($\mathcal{D}(p)$). Because $\mathcal{D}(p)$ is a least fixpoint definition, we do ReInitializeLFD($\mathcal{D}(q)$) in Line 6 of Procedure ReInitializeBelow($\mathcal{D}(p)$). Since $\mathcal{D}(q)$ is a greatest fixpoint definition and $M(q) = \mathbf{f}$, we set $M(q)$ to be true in Line 4 of Procedure ReInitializeLFD($\mathcal{D}(q)$). By the end of this step, we obtain that $M = \{p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$ and $\mathcal{Q} = \{q\}$.*
- *Step 4: Pop p from $\mathcal{Q}$ in Line 22. $Derived(p) = \mathbf{t}$ in Line 24 and thus, $M(q) = Derived(q) = \mathbf{t}$. We go back to the While loop and since $\mathcal{Q}$ is empty, we reach the fixpoint of $\mathcal{D}$. Thus, the resulting model $M = \{p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$ is the desired fixpoint model of $\mathcal{D}$.*

Having this algorithm for computing a model of a propositional fixpoint definition, we can give a simple complexity result about the SAT(FD) problem.

Proposition 6.3.6 *SAT(FD) is* NP*-complete.*

Proof. (Membership) A model of a propositional fixpoint definition can be computed in polynomial time, e.g., using the above algorithm or Browne et

al.'s algorithm. It is easy to define an algorithm that finds SAT(FD) models in polynomial time on a non-deterministic Turing machine.

(Hardness) Any SAT problem is trivially also a SAT(FD) problem, and SAT itself is NP-complete.

6.4 Justification semantics of fixpoint definitions

In this section we perform a semantical analysis of PC(FD), and more specifically, of propositional fixpoint definitions. The central question is: when is a given interpretation a model of a propositional fixpoint definition?

We give a graph theoretical characterization of propositional fixpoint definitions; it uses graph structures called *justifications*. We introduce *justification semantics*, and define some derived concepts useful for SAT(FD) algorithms. Such a semantical analysis is important in its own right, because it yields new insights into the logic. Importantly, this result also forms the theoretical underpinning to develop algorithms for solving the SAT(FD) problem. The results in this section are built on concepts first defined in [52].

6.4.1 Justifications

The truth values of defined symbols are determined by the truth values of open symbols. We want to identify, for each defined literal, what would constitute sufficient reason for it to be either true or false. Consider an atom c defined by the rule $c \leftarrow c_1 \wedge \ldots \wedge c_N$: in order for c to be true, all of $\{c_1, \ldots, c_N\}$ should be true; in order for c to be false, it suffices that one c_i be false. On the other hand, in order for an atom d defined by the rule $d \leftarrow d_1 \vee \ldots \vee d_N$ to be true, it suffices that one d_i be true; in order for d to be false, all of $\{d_1, \ldots, d_N\}$ should be false. We call such direct (local) dependencies $c \rightsquigarrow \{c_1, \ldots, c_N\}$, $d \rightsquigarrow d_i$ *direct justifications*.

Definition 6.4.1 (Direct justification) *Let $l \in \widehat{Def(\mathcal{D})}$. A* direct justification for l*, denoted $DJ(l)$, is a set of literals such that:*

- *if $l \in \mathbf{D}_{lits}$, $DJ(l)$ is a singleton $\{l'\}$, for $l' \in \varphi_l$;*
- *if $l \in \mathbf{C}_{lits}$, $DJ(l) = \varphi_l$.*

However, such direct justifications on their own do not suffice yet to capture the deterministic relationship between open and defined symbols imposed by a fixpoint definition. To capture this relationship we may use a graph structure,

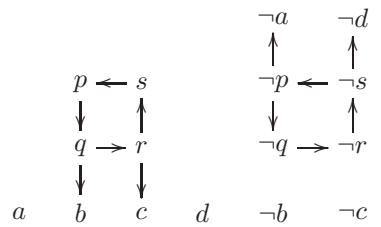

Figure 6.2: A justification J_1 for $\mathcal{D}$

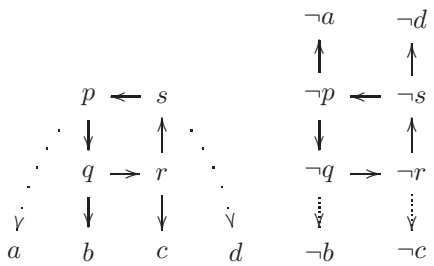

Figure 6.3: Overview of possible justifications for $\mathcal{D}$

built out of direct dependencies as above. This motivates the following definition:

Definition 6.4.2 (Justification) *A* justification J *for* $\mathcal{D}$ *is a directed graph* (V, E) *where* $V = \widehat{\Sigma}$*, and* E *is such that for each* $l \in \widehat{Def(\mathcal{D})}$*, the set* $\{l' \mid (l, l') \in E\}$ *is a direct justification for* l*, denoted* $DJ_J(l)$*, and for each* $l \in \widehat{Open(\mathcal{D})}$*,* E *contains no edge leaving* l*.*

Observe that a justification J is uniquely determined by the function $d_J : \mathbf{D}_{\text{lits}} \to \widehat{\Sigma}$ defined by $d_J(l) = x$ if $DJ_J(l) = \{x\}$, for each $l \in \mathbf{D}_{\text{lits}}$.

Example 6.4.3 *Consider the fixpoint definition* $\mathcal{D} =$

$$\left\lfloor \begin{array}{l} p \leftarrow q \vee a \\ q \leftarrow r \wedge b \\ \left\lceil \begin{array}{l} r \leftarrow s \wedge c \\ s \leftarrow p \vee d \end{array} \right\rceil \end{array} \right\rfloor .$$

Then Figure 6.2 shows a justification, J_1 *for* $\mathcal{D}$*.* J_1 *is characterized by* $d_{J_1}(p) = q, d_{J_1}(s) = p, d_{J_1}(\neg q) = \neg r$*, and* $d_{J_1}(\neg r) = \neg s$*.*

Figure 6.3 shows an overview of all possible justifications for $\mathcal{D}$, whereby possible edges are shown as dotted arrows. For each literal from which possible edges leave, exactly one of the edges must be chosen in an actual justification.

The intuitions given for the concept of justification also motivate the definition of the *support* property.

Definition 6.4.4 (Support) *Let J be a justification for $\mathcal{D}$, I a Σ-interpretation. Then J* supports *I if for each $l \in \widehat{Def(\mathcal{D})}, I(l) = I(\bigwedge DJ_J(l))$.*

The conjunction $\bigwedge DJ_J(l)$ is true in I if all literals in the direct justification of l in J are true. If that conjunction is true, then l itself must be true.

Example 6.4.5 *Example 6.4.3 continued. Consider the interpretation $I_1 = \{p \mapsto \mathbf{t}, q \mapsto \mathbf{t}, r \mapsto \mathbf{t}, s \mapsto \mathbf{t}, a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}, d \mapsto \mathbf{f}\}$. The justification J_1 shown in Figure 6.2 supports I_1.*

Let J be a justification and L a loop in J. L is a set of literals. We call L *positive*, *negative*, or *mixed*, depending on whether it consists respectively of atoms only, of negative literals only, or of both. Note that a loop in a justification of a fixpoint definition $\mathcal{D}$ is either positive or negative due to the fact that all defined atoms occur only positively in each rule body of $\mathcal{D}$.

We partition the set of defined literals $\widehat{Def(\mathcal{D})}$ in two:

$$\mathbf{L}_{\text{lits}} = \{p, \neg p \mid p \text{ is locally defined in a least fixpoint definition}\},$$
$$\mathbf{G}_{\text{lits}} = \{p, \neg p \mid p \text{ is locally defined in a greatest fixpoint definition}\}.$$

Let $\mathcal{D}$ be a propositional fixpoint definition and J a justification for $\mathcal{D}$. It is easy to see that for each loop L in J, there exists a unique subdefinition $\mathcal{D}'$ of $\mathcal{D}$ (possibly $\mathcal{D}' = \mathcal{D}$) such that all literals occurring in L are defined in $\mathcal{D}'$ and at least one literal in L is locally defined in $\mathcal{D}'$. We call such a loop L *$\mathcal{D}'$-loop*. A $\mathcal{D}'$-loop L is called a *least loop*, respectively, a *greatest loop*, if $\mathcal{D}'$ is a least fixpoint definition, respectively, a greatest fixpoint definition. It can be easily shown that for each loop L in J, L is either a least loop or a greatest loop.

Example 6.4.6 *Examples 6.4.3, 6.4.5 continued. The loops in J_1 are $\{p, q, r, s\}$ and $\{\neg p, \neg q, \neg r, \neg s\}$. Consider the loop $L_1 = \{p, q, r, s\}$. L_1 is a positive loop because it consists of atoms only. All atoms in L_1 are defined in $\mathcal{D}$ and both p and q are locally defined in $\mathcal{D}$, hence L_1 is a least loop. Consider instead the loop $L_2 = \{\neg p, \neg q, \neg r, \neg s\}$. L_2 is a negative loop because all literals in L_2 are negative. All literals in L_2 are defined in $\mathcal{D}$ and both $\neg p$ and $\neg q$ are locally defined in $\mathcal{D}$. We therefore have that L_2 is a negative least loop.*

Still building on the intuitions behind the concept of justification, we point out that our explanation contains a causal component: a defined literal l is true because all literals in $\{l' \mid (l, l') \in J\}$ are true, for a given justification. This implies that loops in a justification graph are unwanted: they reflect a cyclic, and therefore flawed, chain of reasons. However, it can be seen from the fixpoint semantics that some sets of literals can cause each other to be false. This motivates the definition of the *loop-safeness* property.

Definition 6.4.7 (Dangerous loop) *We call a loop L in a justification J* dangerous *in interpretation I if either it is a positive least loop in J and contains a true atom in I, or it is a negative greatest loop in J and contains a true negative literal in I.*

Definition 6.4.8 (Loop-safeness) *Let J be a justification for $\mathcal{D}$ and I a Σ-interpretation. Then J is* loop-safe in *I if it contains no dangerous loops in I. I.e.,*

- *for any positive least loop in J, all atoms in the loop are false in I, and*
- *for any negative greatest loop in J, all negative literals in the loop are false in I.*

Example 6.4.9 *Examples 6.4.3, 6.4.5, 6.4.6 continued. J_1 is not loop-safe in I_1, since $\{p, q, r, s\}$ is a positive least loop in J and all atoms in the loop are true in I. Note that the loop $\{\neg p, \neg q, \neg r, \neg s\}$ is harmless: it is a negative least loop.*

Naturally, we are interested in the combination of support and loop-safeness.

Definition 6.4.10 (Witness) *Let J be a justification for $\mathcal{D}$, I a Σ-interpretation. Then J is a* $\mathcal{D}$-witness for *I iff J supports I and J is loop-safe in I. If $\mathcal{D}$ is clear from the context, we simply say J is a witness.*

Example 6.4.11 *Examples 6.4.3, 6.4.5, 6.4.6, 6.4.9 continued. There cannot be a $\mathcal{D}$-witness J for I_1: if the support property is not to be violated, then $d_J(p) = q$ and $d_J(s) = p$, but then $\{p, q, r, s\}$ is a positive least loop in J; if the loop-safeness property is not violated, all atoms in the positive least loop must be false, which is not the case in I_1.*

J_1 in Figure 6.2 is a witness for $I_2 = \{p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{f}, s \mapsto \mathbf{f}, a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}, d \mapsto \mathbf{f}\}$.

With the concept of witness in place, we come to our main result. This result formalizes our intuitions: if there exists a justification that satisfies all criteria it

should—support and loop-safeness—to represent a sound set of reasons for defined literals to be true in a given interpretation, i.e., if there exists a witness for the interpretation, it must be a model of the definition, and vice versa.

Theorem 6.4.12 *Let I be a Σ-interpretation and $\mathcal{D}$ a fixpoint definition on Σ. Then $I \models \mathcal{D}$ iff there exists a $\mathcal{D}$-witness for I.*

Example 6.4.13 *Examples 6.4.3, 6.4.5, 6.4.9, 6.4.11 continued.* $I_2 = \{p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, r \mapsto \mathbf{f}, s \mapsto \mathbf{f}, a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}, d \mapsto \mathbf{f}\}$ *is a model of $\mathcal{D}$.*

We prove this result in the Section 6.4.2, where we first introduce some preliminary results about justifications.

6.4.2 Properties of justifications

We start by some results regarding the *support* property.

Proposition 6.4.14 *Let I be a Σ-interpretation and $\mathcal{D}$ a fixpoint definition over Σ. Then $I \models comp(\mathcal{D})$ iff there exists a justification J for $\mathcal{D}$ that supports I.*

Proof. We do a case analysis on the two kinds of rules of $\mathcal{D}$.

- $p \leftarrow p_1 \vee \ldots \vee p_N \in \mathcal{D}$. If $I \models comp(\mathcal{D})$, we have $I \models p \equiv p_1 \vee \ldots \vee p_N$. Suppose $I(p) = \mathbf{t}$. Then at least one p_i is true in I. Then let $d_J(p) = p_i$: the support condition is thereby satisfied in p. Also, $\neg p$ is then false: the condition is then also satisfied in $\neg p$. Suppose instead $I(p) = \mathbf{f}$. Then all p_i are also false in I. Hence, the support condition is satisfied in p no matter which p_i is chosen as $d_J(p)$. Since all $\neg p_i$ are true in I, also $I(\neg p) = I(\bigwedge_i \neg p_i)$ is satisfied.

  The reverse part is also easy to verify (if the support condition is satisfied in p, then $I \models p \equiv p_1 \vee \ldots \vee p_N$).

- $p \leftarrow p_1 \wedge \ldots \wedge p_N \in \mathcal{D}$. This is symmetric to the previous case.

The following concept, derived from justifications, will be useful in the following text.

Definition 6.4.15 *Let J be a justification for $\mathcal{D}$ and $l \in \widehat{Def(\mathcal{D})}$. The* subjustification *of J starting in l, denoted $Sub(J, l)$, is the subgraph of J consisting of l and all literals reachable from l in J.*

Proposition 6.4.16 *Let J be a justification for $\mathcal{D}$, I a Σ-interpretation, and let J support I.*

1. *Let $l \in \widehat{Def(\mathcal{D})}$ be a literal with $I(l) = \mathbf{t}$. Then all literals in $Sub(J, l)$ are true in I.*
2. *Let L be a loop in J. Then L consists entirely of false, or entirely of true literals in I.*

Proof. We prove item 1 by induction: l is true in I, and if all literals reachable from l in J in i steps are true, then all literals reachable from l in J in $i+1$ steps must be true. Indeed: let l' be a literal reachable from l in J in i steps, and l' is true. Since J supports I, it follows that all literals in $DJ_J(l')$ are true in I. Since all literals reachable from l in J in $i+1$ steps are member of $DJ_J(l')$ for some l' reachable from l in J in i steps, this prove the induction step.

It follows from the definition of $Sub(J, l)$ that a loop L is a subset of $Sub(J, l)$ for any $l \in L$. Hence item 2 is an easy consequence of item 1.

In order to prove Theorem 6.4.12, we introduce some auxiliary concepts and notations as follows.

Let l be a defined literal of $\mathcal{D}$. We call l a *min-literal* of $\mathcal{D}$ if l is an atom and $\mathcal{D}$ is an LFD, or l is a negative literal and $\mathcal{D}$ is a GFD. Otherwise, we call l a *max-literal* of $\mathcal{D}$. Note that the same literal may be min- and max-literal of several definitions in which it is defined.

Recall that given a justification J of $\mathcal{D}$, and $\mathcal{D}'$ a subdefinition of $\mathcal{D}$, possibly $\mathcal{D}$ itself, a $\mathcal{D}'$-loop L of J is a loop of J such that all literals of L are defined in $\mathcal{D}'$ and at least one is locally defined in $\mathcal{D}'$.

Definition 6.4.17 *We call a loop L in a justification J of $\mathcal{D}$* notable *if L is a $\mathcal{D}'$-loop consisting of min-literals of $\mathcal{D}'$, where $\mathcal{D}'$ is a subdefinition of $\mathcal{D}$ or $\mathcal{D}$ itself.*

It can be easily shown that a notable loop is either a positive least loop or a negative greatest loop. Thus, J is loop-safe in I if each notable loop of J contains only false literals in I.

Definition 6.4.18 *Let J be a justification of $\mathcal{D}$. J is* loop-free *if J does not contain notable loops.*

Proposition 6.4.19 *Let J be a $\mathcal{D}$-witness for I. Then for each defined literal $l \in \widehat{Def(\mathcal{D})}$ with $I(l) = \mathbf{t}$, $Sub(J, l)$ is loop-free and contains only true open literals in I.*

Proof. Since J supports I, by Proposition 6.4.16 we obtain that all literals in $Sub(J, l)$ are true, including the open ones. For each defined literal $l \in \widehat{Def(\mathcal{D})}$ with $I(l) = \mathbf{t}$, $Sub(J, l)$ cannot contain a notable loop. Otherwise, by Proposition 6.4.16, this would be a notable loop in J consisting of true literals in I, which contradicts to the fact that J is loop-safe in I.

Definition 6.4.20 *Let J be a justification of $\mathcal{D}$. For each defined min-literal l of $\mathcal{D}$ we define n_l as the the maximal number of literals locally defined in $\mathcal{D}$ on a branch of $Sub(J, l)$.*

Observe that this number n_l can be infinite, in which case J contains a notable loop. On the other hand, n_l is finite if $Sub(J, l)$ is loop-free.

We are now ready to prove Theorem 6.4.12, which states that $I \models \mathcal{D}$ iff there exists a $\mathcal{D}$-witness for I.

Lemma 6.4.21 *Let $\mathcal{D}$ be a propositional fixpoint definition over Σ, J a justification for $\mathcal{D}$ and I_O an Open($\mathcal{D}$)-interpretation. Then for each defined literal $l \in \widehat{Def(\mathcal{D})}$ such that $Sub(J, l)$ is loop-free and contains only true open literals of $\mathcal{D}$, l is true in the model I of $\mathcal{D}$ extending I_O.*

Proof. We prove it by nested induction, first on the structure of $\mathcal{D}$ and then on the number of iterations of $\Gamma_{\mathcal{D}}^{I_O}$.

In the top-level induction, we first need to prove that the lemma holds for unnested definitions, and then it holds for increasing nesting depths. For the induction case, we have to prove that given that the lemma holds for all subdefinitions of $\mathcal{D}$, it holds for $\mathcal{D}$. Notice that the condition of this lemma trivially holds for unnested definitions. Therefore, the proof of the base case is just a special, simplified case of the induction case. So it suffices to prove the induction case.

Assume that the lemma holds for all subdefinitions of $\mathcal{D}$. We obtain the model of $\mathcal{D}$ extending I_O by iterating $\Gamma_{\mathcal{D}}^{I_O}$. This iteration constructs two series of interpretations $I^0, \ldots, I^i, \ldots, I^n$, and $\bar{I}^0, \ldots, \bar{I}^i, \ldots, \bar{I}^n$ where I^0 makes all locally defined min-literals of $\mathcal{D}$ false, and $I^i = \Gamma_D^{I_O}(I^{i-1})$ and $\bar{I}^i$ is the extension of $I_O + I^i$ satisfying all subdefinitions of $\mathcal{D}$.

By the induction hypothesis, if a literal l' is defined in a strict subdefinition $\mathcal{D}'$ of $\mathcal{D}$, then its $\mathcal{D}'$-justification $Sub(J, l')|_{\mathcal{D}'}$, obtained by cutting $Sub(J, l')$ below

the first occurrence of literals locally defined in $\mathcal{D}$, is loop-free. Therefore, if it contains only true leaves in $I_O + I^{i-1}$, then l' is true in $\bar{I}^{i-1}$.

We have the following induction hypothesis IH(i) which is:

If l is a min-literal of $\mathcal{D}$ and $n_l \leq i$, then l is true in $\bar{I}^i$. If l is a max-literal of $\mathcal{D}$ then l is true in $\bar{I}^i$.

Recall that l is a defined literal of $\mathcal{D}$ such that $Sub(J, l)$ is loop-free and contains only true open literals of $\mathcal{D}$ in I_O.

We have to prove the base case, IH(0) and the induction case, that IH($i-1$) implies IH(i).

- (Base case) IH(0).
  - Assume that l is a min-literal of $\mathcal{D}$.
    - l is locally defined in $\mathcal{D}$, trivially.
    - l is not locally defined in $\mathcal{D}$ (this case does not occur when $\mathcal{D}$ is unnested). Suppose that l is defined in the subdefinition $\mathcal{D}'$ of $\mathcal{D}$. Consider the justification $Sub(J, l)$ as a justification for $\mathcal{D}'$. Since $Sub(J, l)$ contains no literals locally defined in $\mathcal{D}$, it holds that each open leaf l' of the justification $Sub(J, l)$ is an open literal of $\mathcal{D}$. It follows directly that l' is an open leaf of J and hence, it is true in I_O. Since $\bar{I}^0$ is a model of $\mathcal{D}'$, it is obtained that l is true in $\bar{I}^0$.
  - Assume that l is a max-literal of $\mathcal{D}$. The proof is quite similar.
    - l is locally defined in $\mathcal{D}$, trivially.
    - l is not locally defined in $\mathcal{D}$ (this case does not occur when $\mathcal{D}$ is unnested). Suppose that l is defined in the subdefinition $\mathcal{D}'$ of $\mathcal{D}$. Consider the justification $Sub(J, l)$ as a justification for $\mathcal{D}'$. An open leaf l' of this justification for $\mathcal{D}'$ is either an open literal of $\mathcal{D}$ or locally defined in $\mathcal{D}$. In the first case, l' is a leaf of J and hence, true. In the second case, it is trivial to show that l' is true. Since $\bar{I}^0 \models \mathcal{D}'$, it follows that l is true in $\bar{I}^0$.
- (Induction case) Assume that IH($i-1$). We now prove IH(i).
  - (a) Assume that l is a min-literal of $\mathcal{D}$.
    - (a.1) l is locally defined in $\mathcal{D}$. If $n_l > i$, nothing is to prove. Suppose $n_l \leq i$ and consider $DJ_J(l)$.
      Each $l' \in DJ_J(l)$ has the property that $n_{l'} < n_l \leq i$, since $Sub(J, l')$ is a part of $Sub(J, l)$, all branches of $Sub(J, l)$ contain l, and no branch of $Sub(J, l')$ does.

We thus have that $Sub(J, l')$ is a loop-free $\mathcal{D}$-justification with only true open literals in I_O. By the induction hypothesis IH$(i-1)$, we have that l' is true in $\bar{I}^{i-1}$. This is true for every $l' \in DJ_J(l)$, hence φ_l is true in $\bar{I}^{i-1}$. It follows that l is true in I^i and also in its extension $\bar{I}^i$.

* (a.2) l is not locally defined in $\mathcal{D}$ (this case does not occur when $\mathcal{D}$ is unnested). Suppose that l is defined in the subdefinition $\mathcal{D}'$ of $\mathcal{D}$.
  Again, if $n_l > i$ then nothing is to prove. So assume $n_l \leq i$.
  Consider the justification $Sub(J, l)|_{\mathcal{D}'}$, the restriction of $Sub(J, l)$ obtained by cutting it below the literals defined locally in $\mathcal{D}$. By this construction, $Sub(J, l)|_{\mathcal{D}'}$ is a justification of $\mathcal{D}'$. Moreover, it is loop-free.
  An open leaf l' of this justification is either an open literal of $\mathcal{D}$ or it is a literal defined locally in $\mathcal{D}$. In the first case, l' is an open leaf of J and hence, it is true in I_O. In the second case, $n_{l'} \leq n_l \leq i$, and we showed in (a.1) that l' is true in I^i. Since $\bar{I}^i$ is a model of $\mathcal{D}'$, if follows that l is true in $\bar{I}^i$.

– (b) Assume that l is a max-literal of $\mathcal{D}$. The proof is quite similar.

* (b.1) l is locally defined in $\mathcal{D}$.
  For each $l' \in DJ_J(l)$, $Sub(J, l')$ is a loop-free $\mathcal{D}$-justification with true open literals in I_O. By IH$(i-1)$, we have that $\bar{I}^{i-1}(l') = \mathbf{t}$. It follows that $\bar{I}^{i-1}(\varphi_l) = \mathbf{t}$ and hence $I^i(l) = \bar{I}^i(l) = \mathbf{t}$.
* (b.2) l is not locally defined in $\mathcal{D}$ but in the subdefinition $\mathcal{D}'$ of $\mathcal{D}$ (this case does not occur when $\mathcal{D}$ is unnested).
  Take $Sub(J, l)|_{\mathcal{D}'}$, the restriction of $Sub(J, l)$ to the $\mathcal{D}'$-justification of l. An open leaf l' of this justification is either an open literal of $\mathcal{D}$ or locally defined in $\mathcal{D}$. In the first case, l' is a leaf of J and hence, true. In the second case, we proved in (b.1) that l' is true. Since $\bar{I}^i \models \mathcal{D}'$, it follows that l is true in $\bar{I}^i$.

We obtain that for each i, IH(i) holds. Given that for each min-literal l, n_l is a well-defined finite number, we have that for each min-literal and for each max-literal l with a loop-free justification $Sub(J, l)$ with only true open literals in it, l is true in the model of $\mathcal{D}$ extending I_O. Thus, the lemma holds for $\mathcal{D}$ as well.

The following lemma, which corresponds to the "if" case of Theorem 6.4.12 is an immediate consequence of Proposition 6.4.19 and Lemma 6.4.21.

Lemma 6.4.22 *Let I be a Σ-interpretation. If there exists a $\mathcal{D}$-witness for I, then $I \models \mathcal{D}$.*

Lemma 6.4.23 *Let $\mathcal{D}$ be a propositional fixpoint definition over Σ and I a Σ-interpretation. If $I \models \mathcal{D}$, then there exists a justification J for $\mathcal{D}$ such that for each defined literal $l \in \widehat{Def(\mathcal{D})}$ with $I(l) = \mathbf{t}$, $Sub(J, l)$ is loop-free and contains only true literals in I.*

Proof. We prove it by induction on the structure of $\mathcal{D}$. For the induction case, we need to prove that if the lemma holds for all subdefinitions of $\mathcal{D}$, then it holds for $\mathcal{D}$. Since the condition is trivially satisfied for an unnested definition, the proof of the base case is a special case of the induction case.

Consider $I_O = I|_{Open(\mathcal{D})}$ and the same sequences $I^0, \ldots, I^i, \ldots, I^n$ and $\bar{I}^0, \ldots, \bar{I}^i, \ldots, \bar{I}^n$ as defined in Lemma 6.4.21. We will construct, together with these sequences, a series of $\mathcal{D}$-justifications J^i for which the following statement IH(i) will be proved:

If l is a min-literal locally defined in $\mathcal{D}$ and l is true in $\bar{I}^i$, then $Sub(J^i, l)$ is loop-free and contains only true literals in I.

Once we have proved this, the limit J^n of this sequence contains a justification for all true min-literals of $\mathcal{D}$ in I. We then need an extra step to extend J^n for all true max-literals of $\mathcal{D}$ in I, and for all false defined literals of $\mathcal{D}$ in I.

Let us observe that for each i, each subdefinition $\mathcal{D}'$ of $\mathcal{D}$ is satisfied in $\bar{I}^i$ and hence, by the induction hypothesis, there is a $\mathcal{D}'$-justification $J^{\mathcal{D}'}$ such that for each true defined literal l of $\mathcal{D}'$, $Sub(J^{\mathcal{D}'}, l)$ is loop-free and contains only true leaves.

We first prove IH(0). Consider I^0 and $\bar{I}^0$ as defined before and construct J^0 as follows:

- For each subdefinition $\mathcal{D}'$, select a justification $J^{\mathcal{D}'}$ satisfying the lemma, and define for each min-literal l of $\mathcal{D}$ defined in $\mathcal{D}'$:
$$DJ_{J^0}(l) := DJ_{J^{\mathcal{D}'}}(l).$$
- For each other literal l defined in $\mathcal{D}$, select an arbitrary direct justification $DJ(l)$ for l and define
$$DJ_{J^0}(l) := DJ(l).$$

Basically, J^0 copies the justifications of the true min-literals defined in subdefinitions and is arbitrary elsewhere. Since locally defined literals of $\mathcal{D}$ are false, each true defined literal l in $\bar{I}^0$ is defined in a subdefinition $\mathcal{D}'$ of $\mathcal{D}$. Open leaves of $Sub(J^{\mathcal{D}'}, l)$ are true in $I_O + I^0$, but since I^0 is totally false, they are open literals of

$\mathcal{D}$. It follows that $Sub(J^0, l) = Sub(J^{\mathcal{D}'}, l)$ and hence, it is loop-free and contains only true leaves in I_O. So, IH(0) holds.

Next, assume that IH$(i-1)$ holds. We define J^i as follows:

- If l is locally defined in $\mathcal{D}$ and $I^i(l) = \mathbf{t}$ while $I^{i-1}(l) = \mathbf{f}$, then $\bar{I}^{i-1}(\varphi_l)$ is true. Select a direct justification $DJ(l)$ that is true in $\bar{I}^{i-1}$ and set
$$DJ_{J^i}(l) := DJ(l).$$
- For each subdefinition $\mathcal{D}'$, select a $\mathcal{D}'$-justification $J^{\mathcal{D}'}$ satisfying the lemma. For each l defined in $\mathcal{D}'$ that is true in $\bar{I}^i$ but false in $\bar{I}^{i-1}$, define
$$DJ_{J^i}(l) := DJ_{J^{D'}}(l).$$
- For each literal that is false in $\bar{I}^i$ or was true in $\bar{I}^{i-1}$, define
$$DJ_{J^i}(l) := DJ_{J^{i-1}}(l).$$

Basically, we copy J^{i-1} for all min-literals that retain their truth value, and for those that become true, we copy the justification $J^{\mathcal{D}'}$.

It is easy to see that this construction satisfies a monotonicity property, namely that if l is a min-literal of $\mathcal{D}$ that is true in $\bar{I}^{i-1}$, then $Sub(J^i, l) = Sub(J^{i-1}, l)$. This is because all literals in $Sub(J^{i-1}, l)$ are true as well in J^{i-1}.

To prove IH(i), let l be an arbitrary true min-literal in $\bar{I}^i$. If l was true in $\bar{I}^{i-1}$, then $Sub(J^i, l) = Sub(J^{i-1}, l)$. Hence, $Sub(J^i, l)$ is loop-free and since all literals in $Sub(J^{i-1}, l)$ were true in $\bar{I}^{i-1}$, they are true in $\bar{I}^i$ as well. So assume that l was false in $\bar{I}^{i-1}$. If l is locally defined in $\mathcal{D}$, then $Sub(J^i, l)$ is a concatenation of a true direct justification $DJ(l)$ with, for each $l' \in DJ(l)$, justifications $Sub(J^{i-1}, l')$. It is straightforward to see that $Sub(J^i, l)$ is loop-free and contains only true literals. Finally, assume that l is defined in a subdefinition $\mathcal{D}'$ of $\mathcal{D}$. Basically, the top of $Sub(J^i, l)$ is the justification obtained by cutting $Sub(J^{\mathcal{D}'}, l)$ below literals l' defined in $\mathcal{D}'$ that were true already in $\bar{I}^{i-1}$. Leaves of this justification are either open literals of $\mathcal{D}$, locally defined in $\mathcal{D}$ or defined in $\mathcal{D}'$ but true in $\bar{I}^{i-1}$. All leaves are true. Since $Sub(J^{\mathcal{D}'}, l)$ was loop-free, also this justification is loop-free. Any branch of $Sub(J^i, l)$ is either a branch of this justification or passes through one of its leaves and arrives in J^{i-1}. Hence, it contains no notable loops.

Let J^n be the justification obtained at the fixpoint $\bar{I}^n = I$. It has the property that for each min-literal l true in I, $Sub(J^n, l)$ is loop-free and contains only true literals. We now have to modify this justification so that the same property also holds for max-literals. We define the justification J as follows:

- If l is false or a true min-literal of $\mathcal{D}$, define
$$DJ_J(l) := DJ_{J^n}(l).$$

- By the induction hypothesis, each subdefinition $\mathcal{D}'$ has a justification $J^{\mathcal{D}'}$ with the property that *each* true defined literal l has a loop-free $Sub(J^{\mathcal{D}'}, l)$ consisting of only true literals. For each true max-literal of $\mathcal{D}$ defined in $\mathcal{D}'$, define
$$DJ_J(l) := DJ_{J^{\mathcal{D}'}}(l).$$
- For each true max-literal l defined locally in $\mathcal{D}$, φ_l is true in I. Select an arbitrary true direct justification $DJ(l)$ of l and define:
$$DJ_J(l) := DJ(l).$$

All that is left to verify is that if l is a true max-literal of $\mathcal{D}$, then $Sub(J, l)$ is loop-free and contains only true literals. It is easy to see from the construction that $Sub(J, l)$ indeed contains only true literals. To see that it is loop-free, observe that a notable loop L in it would be a loop of a subdefinition $\mathcal{D}'$, since L consists of max-literals of $\mathcal{D}$. But it follows from the construction of J for true max-literals defined in subdefinitions that each notable loop L of J of a subdefinition $\mathcal{D}'$ would be a notable loop in $J^{\mathcal{D}'}$ and the latter does not contain such loops.

Lemma 6.4.24 *Let $I \models \mathcal{D}$, and let J be a justification for $\mathcal{D}$ such that for each defined literal $l \in \widehat{Def(\mathcal{D})}$ with $I(l) = \mathbf{t}$, $Sub(J, l)$ is loop-free and contains only true open leaves in I. Then J supports I and is loop-safe in I.*

Proof. We first show that J supports I.

- Let l be a defined literal of $\mathcal{D}$ with $I(l) = \mathbf{t}$. $Sub(J, l)$ is loop-free and contains only true open leaves in I. It follows that for each $l' \in DJ_J(l)$, $Sub(J, l')$ is loop-free and contains only true open leaves in I as well, and hence, by Lemma 6.4.21, $I(l') = \mathbf{t}$.
- Let l be a defined literal of $\mathcal{D}$ with $I(l) = \mathbf{f}$. $I(\varphi_l) = \mathbf{f}$ since I is a model of $\mathcal{D}$. Hence, it can be easily shown that $\bigwedge DJ_J(l) = \mathbf{f}$.

Let L be an arbitrary notable loop in J. Then for each defined literal $l \in L$, it holds that L is a notable loop in $Sub(J, l)$. Suppose that L contains a true defined literal l' in I. It follows that $Sub(J, l')$ is loop-free, which contradicts to the fact that L is a notable loop in $Sub(J, l')$. Thus, each notable loop in J contains only false literals, which implies that J is loop-safe in I.

The following lemma, which corresponds to the "only if" case of Theorem 6.4.12 is an immediate consequence of Lemma 6.4.23 and Lemma 6.4.24.

Lemma 6.4.25 *Let I be a Σ-interpretation, and let $I \models \mathcal{D}$. Then there exists a $\mathcal{D}$-witness for I.*

Theorem 6.4.12 is a direct consequence of Lemma 6.4.22 and Lemma 6.4.25.

6.4.3 Three-valued semantics

Theorem 6.4.12 is an interesting result for SAT(FD) solving: it tells us that we can verify that an interpretation I is a model by finding a witness for it, or prove that it is not a model by showing that there exists no witness for it. However, in practical algorithms, we will construct a model: during its construction, we have a three-valued interpretation. We will prefer to reject candidate interpretations before they are two-valued. To this end we extend the result, and the definition of witness, to three-valued semantics.

Definition 6.4.26 (Three-valued support) *Let J be a justification for $\mathcal{D}$, I a tree-valued Σ-interpretation. Then J* three-valuedly supports *I iff, for each $l \in \widehat{Def(\mathcal{D})}$, if $I(\bigwedge DJ_J(l)) \neq \mathbf{u}$ then $I(l) = I(\bigwedge DJ_J(l))$.*

Recall that the truth of the conjunction $\bigwedge DJ_J(l)$ can be seen as a reason for the truth of a defined literal l. The requirement for three-valued support is that for each defined literal l, whenever that conjunction is two-valued in I, l itself must have the same truth in I. It is then possible that a defined literal already has a two-valued truth value, while its body is still three-valued. This state may come up often in practical SAT(FD) algorithms.

Observe that any justification three-valuedly supports the empty interpretation, and that for two-valued interpretations the notions of support and of three-valued support coincide.

Example 6.4.27 *Consider the definition $\mathcal{D} = \left[\begin{array}{l} p \leftarrow a \\ \lfloor\ q \leftarrow \neg a\ \rfloor \end{array}\right]$, and the three-valued interpretation $I = \{a \mapsto \mathbf{u}, p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$. There exists only one justification for $\mathcal{D}$, and it three-valuedly supports I. However, it supports neither $\{a \mapsto \mathbf{f}, p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$ nor $\{a \mapsto \mathbf{t}, p \mapsto \mathbf{t}, q \mapsto \mathbf{t}\}$.*

Definition 6.4.28 (generalized witness) *Let J be a justification for $\mathcal{D}$, I a three-valued Σ-interpretation. Then J is a* witness for *I iff J three-valuedly supports I and J is loop-safe in I.*

This concept generalizes the concept of witness for two-valued interpretations: if I is a two-valued interpretation, then a generalized witness J for (I, I) is a witness for I.

During SAT(FD) solving we may wish to reject three-valued interpretations that cannot be strengthened to a two-valued interpretation that has a witness.

Since the rest of this chapter is concerned with *constructing* models, and therefore mostly with *three-valued* interpretations, we will from now on assume interpretations to be three-valued, unless we explicitly state otherwise. Accordingly, by support we mean three-valued support, and by a witness we mean a generalized witness. We denote an interpretation I either as a (total) function $I : \Sigma \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, or as a partial function $I : \Sigma \to \{\mathbf{t}, \mathbf{f}\}$. The notation will be clear from the context. In the second case, we denote by $dom(I)$ the set of atoms on which I is defined. We say that interpretation I' is an *extension* of I if $dom(I') \subsetneq dom(I')$ and $I'|_{dom(I)} = I$.

6.4.4 Related work

Justification for the well-founded semantics were first defined, albeit in a different form, in [52], and further elaborated in [48]. The works in [115, 117, 118] are the other publications using justifications for the well-founded semantics. To the best of our knowledge, our work is the only ones that uses justifications for fixpoint semantics.

In the context of stable model semantics, similar notions have been defined. For instance, justifications are used in [146] in the context of debugging of Stable logic programs to provide reasons for the truth of an atom in a stable model, and can therefore be used to explain unintended truths.

In a wide context, graph structures that "justify" interpretations have been used for various purposes, such as for Prolog proof verification [152], and for local search for SAT [91].

6.5 SAT(FD) theory

The SAT problem, deciding the satisfiability of propositional logic theories, is a major research theme. An important research direction is to develop SAT solvers for extensions of PC. For example, SAT Modulo Theories (SMT) [138] is the problem of deciding whether a given ground first-order formula is satisfiable with respect to some given background theory. The use of extended languages leads to broader applicability of SAT-like systems, facilitates the modelling of applications, and may substantially reduce the size of encodings.

In this section, we show SAT(FD) solvers can be built by extending SAT solvers with an additional propagation mechanism suitable for reasoning on fixpoint

definitions. The obvious advantage from this approach is that the SAT(FD) solver benefits from any improvement made to the underlying SAT solver. In particular, it has the same performance on pure propositional problems as the underlying solver. A further advantage is that by separating the two propagation mechanisms (one for propositional theories and one for fixpoint definitions), we also strongly simplify the description of a SAT(FD) solver.

SAT solving is the practice of answering the SAT problem. Although other solving techniques exist, the current state of the art solvers are based on DPLL [43] augmented with the *two watched literal scheme* (2WL) and with *clause learning* [134, 163]. The propagation method that drives this search is *unit propagation*, whereby a literal is assigned true if it occurs as the only non-false literal in a clause. We introduce the 2WL scheme in some more detail later in this section; for more (implementation-oriented) details we refer to [134].

Theorem 6.4.12 suggests the following structure for a SAT(FD) algorithm:

- apply the underlying SAT on $comp(\mathcal{D})$ and on the propositional part of the theory,
- maintain a witness for the interpretation found by the underlying SAT solver.

The main procedure of the SAT(FD) algorithm iterates over the following steps until either a solution is found (a total interpretation and a witness for it) or the search space is exhausted.

1. Select a witness J_w for the current interpretation.
2. Use the underlying SAT solver to update the current interpretation by performing unit propagation on $comp(\mathcal{D})$ and on the propositional part of the theory. If no propagation is possible, first an undefined atom is interpreted by $\mathbf{t}$ or $\mathbf{f}$. This is a propagation step by the underlying SAT solver.
3. Use the state of the underlying SAT solver to construct a supporting justification J_s.
4. If J_s is not a witness for the current interpretation then:
   (a) Use J_w to adjust J_s so that it becomes a witness. In case this fails, a set U is obtained of defined literals that cannot have a witness under the current interpretation, i.e., the search for the witness is unable to find a good justification for the literals in U.
   (b) If J_w is still not a witness for the current interpretation, make sure all literals in U will be set to false in the next iteration, e.g., by extending the theory with *reason clauses*.

r	$d \leftarrow d_1 \vee \ldots \vee d_N$	$c \leftarrow c_1 \wedge \ldots \wedge c_N$
$B_1(r)$	$d \vee \neg d_1$	$\neg c \vee c_1$
$\vdots$	$\vdots$	$\vdots$
$B_N(r)$	$d \vee \neg d_N$	$\neg c \vee c_N$
$C(r)$	$\neg d \vee d_1 \vee \ldots \vee d_N$	$c \vee \neg c_1 \vee \ldots \vee \neg c_N$

Figure 6.4: Clauses in the CNF representation of a fixpoint definition's completion

Example 6.5.1 *Let* $\mathcal{D} = \left\lfloor \begin{array}{l} p \leftarrow q \\ \lceil \ q \leftarrow p \vee r \ \rceil \end{array} \right\rfloor$,

and let the current interpretation have r *false, and* p *and* q *unknown. A* $\mathcal{D}$*-witness* J_w *for the previous interpretation, which interprets unknown to all atoms* p, q *and* r*, has* $d_{J_w}(p) = r$ *but is not supporting in the current interpretation. A supporting justification for* $\mathcal{D}$ *has* $d_{J_s}(p) = q$ *but has a positive least loop consisting of unknown atoms in the current interpretation, which means that* J_s *is not loop-safe and hence, it is not a witness under the current interpretation. Adjusting* $d_{J_s}(p)$ *such that it becomes a witness under the current interpretation is not possible; the adjustment fails and the set* $\{p, q\}$ *is returned. In the next iteration, unit propagation will set* p *and* q *to false; the witness justification* J_w *becomes a supporting one and a solution is obtained.*

Step 1: Selection of a witness. In the first iteration, the witness J_w is constructed by applying step 2, step 3 and step 4, which are illustrated as below, except that we make a small change in step 4. In later iterations, the most recent witness is used as the desired justification J_w. Taking the most recent witness keeps the difference with the supporting justification to be constructed in step 3 to a minimum and, as will become clear, this reduces the amount of work in step 4(a).

Step 2: SAT solving. This is a propagation step by the underlying SAT solver. Note that this includes clause learning and backtracking when propagation leads to the detection of a conflict.

Contemporary SAT solvers have *unit propagation* as their propagation mechanism: whenever in an input clause $\varphi = l_1 \vee \ldots \vee l_n$ all l_i are false, except one, this so-called *unit literal* is made true.

We can easily apply unit propagation on *comp*($\mathcal{D}$) and the propositional part of the theory by means of CNF unit propagation. All that is required is to represent

comp($\mathcal{D}$) in CNF. Figure 6.4 gives the set of clauses $\{C(r), B_1(r), \ldots, B_N(r)\}$ derived from disjunctive and conjunctive rules $r \in \mathcal{D}$, which is the CNF representation of *comp*($\mathcal{D}$). We call the clauses denoted by $C(r)$ in Figure 6.4 *long clauses*.

One particularly effective algorithm for CNF unit propagation is the "two watched literals schema" (2WL schema), introduced in [134]. In any clause φ with at least two literals, two different literals $w_1(\varphi)$ and $w_2(\varphi)$ are denoted as "watched". Then the algorithm ensures that the following invariant remains satisfied: $\forall \varphi \in \Psi : I(w_1(\varphi) \wedge w_2(\varphi)) \neq \mathbf{f} \vee I(w_1(\varphi) \vee w_2(\varphi)) = \mathbf{t}$, where I is the current interpretation. To implement this, each literal keeps a list of the clauses in which it is watched. If then a literal becomes false, those clauses are visited, and we have the following possibility for each clause φ:

- there is a third literal $l \in \varphi$ such that $I(l) = \mathbf{t}$: the watch from the false literal can move to l;
- there is no such third literal: then the *other* watched literal must be true, if φ is to be satisfied. It may already be true. It may also already be false; in that case a conflict is derived. This last case may arise because literals that obtain a two-valued truth value must be processed sequentially: when the two watches of φ both become false, one of them will necessarily be processed first.

Step 3: Constructing a supporting justification J_s. We assume here that the underlying SAT solver we are extending implements unit propagation using the 2WL scheme. A justification J_s that supports the current interpretation I can simply be constructed. This requires to set $d_{J_s}(l)$ for every $l \in \mathbf{D}_{\text{lits}}$. Let $d \leftarrow d_1 \vee \ldots \vee d_N$ be the rule defining d in $\mathcal{D}$. *comp*($\mathcal{D}$) contains the long clause $\neg d \vee d_1 \vee \ldots \vee d_N$; let w_1 and w_2 be the watched literals of this clause. If $w_1 = \neg d$ then $d_{J_s}(d) = w_2$, otherwise, $d_{J_s}(d) = w_1$. Let $c \leftarrow c_1 \wedge \ldots \wedge c_N$ be the rule defining c in $\mathcal{D}$. *comp*($\mathcal{D}$) contains the long clause $c \vee \neg c_1 \vee \ldots \vee c_N$; let w_1 and w_2 be the watched literals of this clause. If $w_1 = c$ then $d_{J_s}(\neg c) = w_2$, otherwise, $d_{J_s}(\neg c) = w_1$. Knowing that the current interpretation I is the result of a propagation step, it can be easily verified that J_s is a supporting justification with respect to I.

Step 4(a): Find a witness. The supporting justification J_s can have dangerous loops. Because J_w is loop-safe, it must be the case that each dangerous loop in J_s contains at least one $l \in \mathbf{D}_{\text{lits}}$ with $d_{J_s}(l) \neq d_{J_w}(l)$. Let us call such literals *cycle sources*. Cycle sources belong to the set of literals on which both justifications disagree: $CS = \{l \in \mathbf{D}_{\text{lits}} \mid d_{J_s}(l) \neq d_{J_w}(l)\}$. The overall strategy is to check for each element *cs* in CS whether it is a cycle source, and, if so, to perform *local*

adjustments on the supporting justification J_s so that cs is no longer part of a dangerous loop ("justifying cs"). Obviously, the smaller CS, the less work this step requires.

These local adjustments may or may not update $d_{J_s}(cs)$ and may update other elements in which J_s does not differ from J_w . The point is that updates break up the dangerous loops passing through cs. Hence, if all elements of CS can be successfully processed, J_s becomes loop-safe, i.e., a witness of the current interpretation.

For what concerns the elements l in CS that are false under the current interpretation, one can observe that, due to the propagation, all body literals in φ_l must be false as well. Hence for such l, J_s remains supporting when setting $d_{J_s}(l) = d_{J_w}(l)$ and these elements can be removed from CS. The further processing then consists of justifying each element cs in CS until either CS is empty and hence J_s is a witness of the current interpretation or some cs could not be justified, in which case a set U as described in the high level algorithm is returned.

Step 4(b): Learning clauses from the U-set. When the supporting justification cannot be adjusted into a loop-safe supporting justification, the set U of defined literals that cannot have a witness subjustification is returned. These defined literals have to be set to false in the current interpretation as they necessarily have to be false in each fixpoint model extending the current interpretation. To properly integrate this with the SAT solver and its backtracking search, this is achieved by extending the theory with an appropriate *learned clause* for each of these literals [163].

Recall that U's external disjuncts is defined as $(\bigcup_{l \in \mathbf{D}_{\text{lits}} \cap U} \varphi_l) \setminus U$ i.e.: all body literals of the disjunctively defined literals in U except the literals in U themselves. The falsity of those literals forces the falsity of the literals in U. A so-called *loop formula* $\bigvee U \supset \bigvee \mathbf{D}^{\text{ext}}(U)$ captures this (adapted from [106]); its CNF contains one reason clause for each literal in U.

6.6 Algorithms

In this section we propose concrete algorithms for SAT(FD).

6.6.1 UnitPropDef

An algorithm UnitPropDef for applying unit propagation on a PC(FD) theory, which is in ECNF, is easy to construct by means of an algorithm for CNF unit propagation. All that is required is to represent $comp(\mathcal{D})$ in CNF form (cf. Figure 6.4; observe in particular that this representation is linear in the size of $\mathcal{D}$), and then apply the algorithm for CNF unit propagation.

It is easy to verify that CNF unit propagation on the clauses of $comp(\mathcal{D})$ implements UnitPropDef.

Remark 6.6.1 *Some ASP solvers provide four different propagation mechanisms for rules (from head to body and vice versa, for both disjunctive and conjunctive rules), which together are equivalent to the* UnitPropDef *rule. For instance,* SMODELS *implements these four mechanisms in its* AtLeast *procedure [166], using a counter-based algorithm derived from [58], which is the algorithm for computing the deductive closure of a set of rules.*

6.6.2 An improved algorithm for the computation of fixpoint definitions

In this section, we propose an alternative algorithm for the computation of fixpoint definitions, which simulates Browne et al.'s standard algorithm as Algorithm 1. The basic idea behind the algorithm in this section is same as that of the standard algorithm for the computation of fixpoint definitions in Section 6.3.4 except that a justification is created during the computation of a fixpoint definition.

Recall some concepts and notations which will be used to present the concrete algorithm.

Let $\mathcal{D}$ be a fixpoint definition, $a \in Def(\mathcal{D})$ and $a \leftarrow \varphi_a$ the rule defining a in $\mathcal{D}$. We use $body(a)$ to denote *the set of literals* in the body of the rule defining a. We denote by $\mathcal{D}(a)$ the subdefinition of $\mathcal{D}$ (or $\mathcal{D}$ itself) in which a is locally defined. We also define the *levels* of $\mathcal{D}$ and each subdefinition of $\mathcal{D}$. This is a characterization of the hierarchy of a fixpoint definition.

Consider a least fixpoint definition $\mathcal{D} = \lfloor\ \mathcal{R}, \Delta_1, \ldots, \Delta_m, \nabla_1, \ldots, \nabla_n\ \rfloor$. The levels of $\mathcal{D}$ and each subdefinition (or, inner fixpoint definition) of $\mathcal{D}$ are defined by:

1. The level of $\mathcal{D}$ is 1.
2. Let $\mathcal{D}'$ be a subdefinition of $\mathcal{D}$, which is either a least fixpoint definition of the form $\lfloor\ \mathcal{R}', \Delta'_1, \ldots, \Delta'_m, \nabla'_1, \ldots, \nabla'_n\ \rfloor$ or a greatest fixpoint definition of

the form $\lceil\ \mathcal{R}', \Delta'_1, \ldots, \Delta'_m, \nabla'_1, \ldots, \nabla'_n\ \rceil$, and let the level of $\mathcal{D}'$ be n. Then the level of each Δ'_i is $n+1$ and the level of each ∇'_j is $n+1$ as well.

The level of a greatest fixpoint definition is similarly defined.

The queue $\mathcal{Q}$ used in the algorithm is a stratified First-In-First-Out (FIFO) queue. There is an FIFO queue $\mathcal{Q}_{\mathcal{D}'}$ for each subdefinition $\mathcal{D}'$ of $\mathcal{D}$ (include $\mathcal{D}$ itself) in $\mathcal{Q}$. $\mathcal{Q}$ is stratified by the levels of each subdefinition of $\mathcal{D}$ and $\mathcal{D}$ itself. Dequeuing an atom from $\mathcal{Q}$ always select an atom in a queue $\mathcal{Q}'$ which is in the deepest level of $\mathcal{Q}$. This structure of $\mathcal{Q}$ guarantees that a defined atom a is popped from $\mathcal{Q}$ to be evaluated only when all subdefinitions of $\mathcal{D}(a)$ have already reached fixpoints.

A watched literal mechanism is used in this algorithm. During the computation of fixpoint definitions by applying this algorithm, we use a watched literal to evaluate the truth value of $body(a)$ for each defined atom a. Then we can check whether there is a mismatch between the truth value of a defined atom a and its watched literal in the current interpretation. If such a mismatch occurs, we push this defined atom a to its queue $\mathcal{Q}_{\mathcal{D}(a)}$ in $\mathcal{Q}$. The truth value of a will be adjusted right after all subdefinitions of $\mathcal{D}(a)$ have reached fixpoints and the mismatch between the truth value of a and its watched literal still persists by then.

This watched literal mechanism's strength is that it reduces the number of accesses to other literals and thus, it is more efficient than the standard algorithm in Section 6.3.4. However, we still have to access all disjunctively defined atoms b with a in its body when a becomes true and all conjunctively defined atom b with a in its body when a is made false. Fortunately, the test in these cases is constant time. So compared to the standard algorithm with the counting mechanism in Section 6.3.4, we can only save on half of the work. Therefore, the advantage of the watched literal mechanism might be less dramatic over the counting mechanism than in the case of unit propagation.

For each defined atom a, $W(a)$ is a literal in $body(a)$. If $a \in \mathbf{C}_{\text{lits}}$, $W(a)$ points to a body element of a with the minimal truth value in the current interpretation, i.e., $W(a) = l$ with $l \in body(a)$ and $M(l) = min_{\leq}\{M(l') \mid l' \in body(a)\}$. If $a \in \mathbf{D}_{\text{lits}}$, $W(a)$ points to a body element of a with the maximal truth value in the current interpretation, i.e., $W(a) = l$ with $l \in body(a)$ and $M(l) = max_{\leq}\{M(l') \mid l' \in body(a)\}$. In the above construction of W, it is always better to select an open literal for $W(a)$ if one is available. W determines both a positive justification J^+ and a negative justification J^- as: $J^+(a) = W(a)$ if $a \in \mathbf{D}_{\text{lits}}$ and $J^+(a) = body(a)$ if $a \in \mathbf{C}_{\text{lits}}$, respectively, $J^-(\neg a) = \neg W(a)$ if $\neg a \in \mathbf{D}_{\text{lits}}$ and $J^-(\neg a) = \neg body(a)$ if $\neg a \in \mathbf{C}_{\text{lits}}$. The combination of J^+ with J^- determines a standard justification for $\mathcal{D}$. We call such a justification for $\mathcal{D}$ the justification *created from* W.

Algorithm 4 is the concrete algorithm. The procedure "ReInitializeBelow($\mathcal{D}$)" in it is used to re-initialize all the subdefinitions of $\mathcal{D}$ if there is an update in

Algorithm 4: Improved algorithm with justification

Let $\mathcal{Q} := \emptyset$;
foreach *subdefinition* $\mathcal{D}'$ *of* $\mathcal{D}$ *(include* $\mathcal{D}$ *itself)* **do**
  if $\mathcal{D}'$ *is a greatest fixpoint definition* **then**
    foreach *defined atom* a *which is locally defined in* $\mathcal{D}'$ **do** $M(a) := \mathbf{t}$;
  else
    foreach *defined atom* a *which is locally defined in* $\mathcal{D}'$ **do** $M(a) := \mathbf{f}$;
foreach *defined atom* a **do**
  if $a \in \mathbf{C}_{lits}$ **then**
    Let $W(a) := l$ such that $l \in body(a)$ and $M(l) = min_{\leq}\{M(l') \mid l' \in body(a)\}$;
  else
    Let $W(a) := l$ such that $l \in body(a)$ and $M(l) = max_{\leq}\{M(l') \mid l' \in body(a)\}$;
  if $M(a) \neq M(W(a))$ **then** Add a to $\mathcal{Q}_{\mathcal{D}(a)}$;
while $\mathcal{Q} \neq \emptyset$ **do**
  Pop an atom a from $\mathcal{Q}$;
  if $M(a) = M(W(a))$ **then**
    Skip ;
  else
    Let $M(a) := M(W(a))$;
    if $M(a) = \mathbf{t}$ **then**
      PropagateTrue(a) ;
    else
      PropagateFalse(a) ;
    ReInitializeBelow($\mathcal{D}(a)$)
return M, W

Function PropagateTrue(a)

foreach $b \in \mathbf{C}_{lits}$ *such that* $W(b) = a$ **do**
  if *there is another literal* $l \in body(b)$ *such that* $M(l) = \mathbf{f}$ **then**
    $W(b) = l$;
  else
    Add b to $\mathcal{Q}_{\mathcal{D}(b)}$;
foreach $b \in \mathbf{D}_{lits}$ *such that* $a \in body(b)$ **do**
  if $M(W(b)) \neq \mathbf{t}$ **then**
    Let $W(b) := a$;
    Add b to $\mathcal{Q}_{\mathcal{D}(b)}$;

Function PropagateFalse(a)

```
1 foreach b ∈ D_lits such that W(b) = a do
2     if there is another literal l ∈ body(b) such that M(l) = t then
3         W(b) = l ;
4     else
5         Add b to Q_D(b) ;
6 foreach b ∈ C_lits such that a ∈ body(b) do
7     if M(W(b)) = t then
8         Let W(b) := a ;
9         Add b to Q_D(b) ;
```

Procedure ReInitializeBelow($\mathcal{D}$)

```
1 if D is a greatest fixpoint definition then
2     foreach subdefinition D' of D do
3         ReInitializeGFD(D');
4 else
5     foreach subdefinition D' of D do
6         ReInitializeLFD(D');
```

Procedure ReInitializeLFD($\mathcal{D}$)

```
1 if D is a greatest fixpoint definition then
2     foreach defined atom a which is locally defined in D do
3         if M(a) ≠ t then
4             Let M(a) := t ;
5             PropagateTrue(a) ;
6 foreach definition D' such that D' is a subdefinition of D do
7     ReInitializeLFD(D')
```

Procedure ReInitializeGFD($\mathcal{D}$)

```
1 if D is a least fixpoint definition then
2     foreach defined atom a which is locally defined in D do
3         if M(a) ≠ f then
4             Let M(a) := f ;
5             PropagateFalse(a) ;
6 foreach definition D' such that D' is a subdefinition of D do
7     ReInitializeGFD(D')
```

$\mathcal{D}$ after the iterated induction. The procedure "ReInitializeLFD($\mathcal{D}$)" is used to re-initialize a least fixpoint definition while the procedure "ReInitializeGFD($\mathcal{D}$)" is used to re-initialize a greatest fixpoint definition. According to Browne et al.'s general algorithm, it can be exploited that in case of reiteration due to an update in a least fixpoint definition $\mathcal{D}$, we only need to re-initialize all greatest fixpoint subdefinitions of $\mathcal{D}$, while in case of reiteration due to an update in a greatest fixpoint definition $\mathcal{D}$, we only need to re-initialize all least fixpoint subdefinitions of $\mathcal{D}$.

The following properties can easily be verified to be invariants for the algorithm.

1. $M(body(a)) = M(W(a))$.
2. If there is a mismatch between the truth value of a and its watched literal $W(a)$ in the current interpretation, i.e., $M(W(a)) \neq M(a)$, then a is in the queue $\mathcal{Q}$.

Recall that the structure of $\mathcal{Q}$ guarantees that a defined atom a is dequeued to be evaluated if all subdefinitions of $\mathcal{D}(a)$ have reached fixpoints. With this property of the queue, which is used in the algorithm, and the above invariants, it can be easily shown that when we apply the algorithm on a definition $\mathcal{D}$, each step corresponds exactly to the iteration of the computation of the fixpoint model of $\mathcal{D}$. Thus, the above reasoning and the correctness of Algorithm 1 leads to the correctness of Algorithm 4.

Proposition 6.6.2 *If Algorithm 4, called on a fixpoint definition $\mathcal{D}$, returns M, then M is a fixpoint model of $\mathcal{D}$.*

There is another invariant about loops in the justification J created by Algorithm 4. We first introduce an auxiliary concept called *undesired loop*.

Definition 6.6.3 (Undesired loop) *We call a loop of a justification* undesired *in interpretation I if it is either a positive least loop such that all literals in the loop are true I or a negative greatest loop such that all literals in the loop are true in I.*

Proposition 6.6.4 *If a fixpoint is reached on a definition $\mathcal{D}$ by applying Algorithm 4, J is the justification for $\mathcal{D}$ created by applying Algorithm 4 and M is the resulting model for $\mathcal{D}$, then there are no undesired loops of J in M.*

Proof. We do a case analysis on the two types of $\mathcal{D}$.

- $\mathcal{D}$ is a least fixpoint definition. We prove the property by the induction hypothesis that when an atom a is popped from $\mathcal{Q}$ to be executed, there are no undesired loops in any of $\mathcal{D}(a)$'s subdefinitions with respect to the current interpretation M and the current justification J.
  - Initially, $M(a) = \begin{cases} \mathbf{f} & \text{if } a \text{ is locally defined in a least fixpoint} \\ & \text{subdefinition of } \mathcal{D} \text{ (and } \mathcal{D} \text{ itself)} \\ \mathbf{t} & \text{if } a \text{ is locally defined in a greatest fixpoint} \\ & \text{subdefinition of } \mathcal{D} \end{cases}$.
    It is obvious that there are no undesired loops in the initial interpretation M.
  - Let a be the first element in $\mathcal{Q}_{\mathcal{D}}$ to be executed. It follows that all subdefinitions of $\mathcal{D}(a)$ have reached fixpoints by the structure of $\mathcal{Q}$. Note that $\mathcal{D} = \mathcal{D}(a)$. By the induction hypothesis, it holds that there are no undesired loops in all subdefinitions of $\mathcal{D}$ with respect to the current interpretation M and the current justification J. We can also have that all atoms which are locally defined in $\mathcal{D}$ are false in the current interpretation M. We want to show that executing Algorithm 4 on a will not induce any undesired loops. Because a is locally defined in a least fixpoint definition and it is the first time for a to be popped from $\mathcal{Q}$, we have that $M(a) = \mathbf{f}$ and $M(W(a)) = \mathbf{t}$. Then we set $M(a)$ to be true in Line 18, do PropagateTrue(a) in Line 20 and do ReInitializeBelow($\mathcal{D}(a)$) in Line 23. First, we want to show that executing Algorithm 4 on a does not induce any undesired greatest loops. Actually, since $\mathcal{D}(a)$ is a least fixpoint definition, we have to re-initialize every greatest fixpoint subdefinition of $\mathcal{D}(a)$. It follows that for every atom b which is locally defined in a greatest fixpoint subdefinition of $\mathcal{D}(a)$, $M(b)$ is set to be true after doing ReInitializeLFD procedure on $\mathcal{D}(b)$. Thus, there exists at least one false literals in each negative greatest loop in J, which means that there are no undesired greatest loops, as desired.
    Next, we want to show that executing Algorithm 4 on a does not induce any undesired least loops. We focus on the PropagateTrue(a) procedure.

For each disjunctively defined atom b such that a occurs in $body(b)$ and b is locally defined in a subdefinition of $\mathcal{D}(a)$, $W(b)$ is changed to a if and only if $M(W(b)) = \mathbf{f}$. Such adjustment on $W(b)$ does not induce any undesired least loops because $M(b) = \mathbf{f}$. For each disjunctively defined atom b such that a occurs in $body(b)$ and b is locally defined in $\mathcal{D}$, it holds that $M(b) = \mathbf{f}$ and thus, b is not included in any undesired least loop. It is similar for the case of every conjunctively defined atom b with $W(b) = a$.

Hence, we obtain that executing Algorithm 4 on a does not induce any undesired loops and thus, we complete the proof of the induction case.

- $\mathcal{D}$ is a greatest fixpoint definition. This is symmetric to the previous case.

Proposition 6.6.5 *If Algorithm 4 returns M, W and J is the justification for $\mathcal{D}$ created from W, then J supports M and J is loop-safe to M, i.e., J is a witness for M.*

Proof. It can be easily verified that J supports M by the invariant 2 of this algorithm and the construction of J from W. By the above proposition and the fact that J supports M, we have that J is loop-safe to M .

Algorithm 4 will be applied to make local changes to a given justification in the algorithm for finding local witnesses in the next section.

6.6.3 The algorithm for finding local witnesses

Recall that our strategy will be to search for (construct) a $\mathcal{D}$-witness. When the search fails, there is a set U of defined literals which should be false in each fixpoint model of $\mathcal{D}$ extending the current interpretation M: instead of finding a $\mathcal{D}$-witness, the search will then return U.

We construct a $\mathcal{D}$-witness by making *local* adjustments to a given supporting justification. We first define a helpful concept.

Definition 6.6.6 (Local witness) *Let M be a (three-valued) interpretation, $l \in \widehat{Def(\mathcal{D})}$ a defined literal with $M(l) \neq \mathbf{f}$, and J a justification for $\mathcal{D}$. Then J is a* local $\mathcal{D}$-witness for M in l *if J supports M, J contains no positive least loops that contain l, and J contains no negative greatest loops that contain l.*

A justification J that is a local witness for M in all non-false literals $l \in \widehat{Def(\mathcal{D})}$ is a witness for M. On the other hand, if no local witness exists for M in some literal l, then also there exists no witness for M.

In this section, we propose the algorithm for finding a local witness by making local changes to a given justification. Our algorithm starts from a given interpretation M, a given justification J_s that supports M, and a given literal cs with $M(cs) \neq \mathbf{f}$, called a *cycle source*, and tries to find a local witness for M in cs. It either succeeds, or it finds a set of defined literals which should be false in each fixpoint model of $\mathcal{D}$ extending M.

We first give a brief intuition behind the algorithm. Starting from cs, we first find a set of literals that might contain a loop through cs. Then we try to "justify" literals of that set by making local adjustments to J_s. A literal is "justified" once it is certain that J_s contains no path from this literal to cs in any positive least loop or any negative greatest loop. When cs itself is justified, J_s has neither positive least loop nor negative greatest loop through it, and because it supports M, it is then a local witness for M in cs.

We now give a more comprehensive high-level overview of this algorithm.

- Initially, the algorithm constructs the set HP ("has path") of all literals that have a path (of length at least one) to cs in J_s, such that no literal along the path is false in M. This is a simple depth-first search in the transpose of J_s (i.e., the symmetric graph of J_s). Hence, all literals belonging to eventual dangerous loops in J_s that contain cs are in HP, and possibly some more. However, if $cs \notin HP$, then J_s is already a local witness for M in cs.
- The algorithm attempts to "justify" literals in HP by applying Algorithm 4 on the restriction of $\mathcal{D}$ to HP, denoted by $\mathcal{D}|_{HP}$. Suppose that Algorithm 4 returns a model M' of $\mathcal{D}|_{HP}$ and W after the execution on $\mathcal{D}|_{HP}$. Let J' be the justification for $\mathcal{D}|_{HP}$ created from W. Let $U = \{l \in HP \mid M'(l) = \mathbf{f}\}$. Consider the following two cases on U.
  - U is empty. By Proposition 6.6.5, it is obtained that J' is a $\mathcal{D}|_{HP}$-witness for M'. Since cs is not false in M' and J' is a $\mathcal{D}|_{HP}$-witness for M', it holds that J' contains neither positive least loops that contain cs nor negative greatest loops that contain cs. Therefore, the algorithm can make local adjustments to J_s by setting $d_{J_s}(l) = d_{J'}(l)$ for each $l \in HP \cap \mathbf{D}_{lits}$, and then cs is justified. The invariant that J_s support M is retained. M itself remains unchanged throughout the algorithm. Since all literals in HP are non-false in M' and J' supports M', it is obtained that $M'(d_{J'}(l')) \neq \mathbf{f}$ for each $l' \in HP \cap \mathbf{D}_{\text{lits}}$. $d_{J'}(l')$ is either in HP or a non-false literal in M. Thus, the modified justification J_s supports M. It follows that J_s is a local $\mathcal{D}$-witness for M in cs.
  - U is not empty. For each $l \in HP \setminus U$, it holds that J' contains neither positive least loops that contain l nor negative greatest loops that contain l. Therefore, the algorithm can make local adjustments to J_s by setting $d_{J_s}(l) = d_{J'}(l)$ for each $l \in (HP \setminus U) \cap \mathbf{D}_{\text{lits}}$. On the

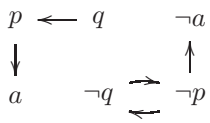

Figure 6.5: Justification with $d_{J_s}(p) = a$

other hand, by the correctness of Algorithm 4, all literals in U should be false in each model of $\mathcal{D}$ extending M. The algorithm terminates and returns the set U. We will add $\bigvee U \supset \bigvee \mathbf{D}^{\text{ext}}(U)$ as the learning clauses to the theory (adapted from [106]).

Algorithm 5 is the concrete algorithm. The above discussion leading to this algorithm can be summarized by the following result:

Algorithm 5: FindLocalWitnesses($cs, M, J_s, \mathcal{D}$)–Justifying a cycle source cs

Result: A set of literals that is either empty or consists of literals which are false in each model of $\mathcal{D}$ extending M. J_s may change; if result is $\emptyset$, J_s is a local $\mathcal{D}$-witness for M in cs.

1 Let $HP := \{l \mid l$ has a path to cs through non-false literals $\}$;
2 **if** $cs \notin HP_J(cs)$ **then return** $\emptyset$;
3 Let $M' :=$ the resulting model of $\mathcal{D}|_{HP}$ of Algorithm 4 on $\mathcal{D}|_{HP}$;
4 Let $W :=$ the result of Algorithm 4 on $\mathcal{D}|_{HP}$;
5 Let $J' :=$ the justification for $\mathcal{D}|_{HP}$ created from W ;
6 Let $U := \{l \in HP \mid M'(l) = \mathbf{f}\}$;
7 **foreach** $l \in (HP \setminus U) \cap \mathbf{D}_{lits}$ **do** $d_{J_s}(l) := d_{J'}(l)$;
8 **return** U ;

Proposition 6.6.7 *Let M be an interpretation, J_s a justification that supports M, and cs a cycle source with $M(cs) \neq \mathbf{f}$. If Algorithm 5, called on cs, returns a non-empty set U, then all literals in U are false in each model of $\mathcal{D}$ extending M. If instead Algorithm 5 returns the empty set, the resulting J_s is a local $\mathcal{D}$-witness for M in cs.*

Example 6.6.8 *Let* $\mathcal{D} =$

$$\left\lfloor \begin{array}{c} p \leftarrow q \vee a \\ \lceil\ q \leftarrow p\ \rceil \end{array} \right\rfloor .$$

- *Let M be the empty interpretation, J_s the justification with $d_{J_s}(p) = a$ (see Figure 6.5), and $cs = p$. Call Algorithm 5 on p. Then $HP = \{q\}$ in Line*

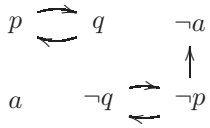

Figure 6.6: Justification with $d_{J_s}(p) = q$

1: q is the only literal that has a non-empty path in J_s to p. Since $p \notin \{q\}$, the algorithm returns the empty set; J_s is a local witness for M in p (and, incidentally, a witness for M).

- *Let M be the empty interpretation as before, $cs = p$, but now $d_{J_s}(p) = q$ (see Figure 6.6). Call Algorithm 5 on p. Then $HP = \{p, q\}$ in Line 1. The algorithm continues with applying Algorithm 4 on $\mathcal{D}|_{\{p,q\}}$, which is $\mathcal{D}$ itself. It finds that M' is the empty interpretation and $d_{J'}(p) = a$ after executing Algorithm 4 on $\mathcal{D}$. Then $U = \emptyset$. Hence it sets $d_{J_s}(p) := d_{J'}(p) = a$ in Line 7. Again the algorithm returns the empty set and (the changed) J_s is a local witness for M in p.*

- *Let M be $\{a \mapsto \mathbf{f}\}$, $cs = p$, and $d_{J_s}(p) = q$. Call Algorithm 5 on p. The algorithm starts the same as in the previous case, finding $HP = \{p, q\}$ and applying Algorithm 4 on $\mathcal{D}$. It finds that $M' = \{p \mapsto \mathbf{f}, q \mapsto \mathbf{f}, a \mapsto \mathbf{f}\}$ in Line 3 and $U = \{p, q\}$ in Line 6. So the algorithm returns the set $U = \{p, q\}$ in Line 8. Indeed, both p and q are false in each model of $\mathcal{D}$ extending $\{a \mapsto \mathbf{f}\}$.*

We give the algorithm for constructing a witness for M from local witnesses for M in various cycle sources in next section.

6.6.4 The algorithm for finding witnesses

The search for a $\mathcal{D}$-witness for M is guided by the concept of *cycle source*: the literal where the search begins. We want the property that justifying a given cycle source actually leads to a $\mathcal{D}$-witness in M to be as high as possible, in other words, we want as few cycle sources as possible. On the other hand, we also want to have a complete algorithm, hence, for the set U of literals which should be false in each fixpoint model of $\mathcal{D}$ extending M there should be at least one cycle sources $cs \in U$.

We take into account previous searches for witnesses in M. This also requires an *initial* search, which will be illustrated later in this section. Let M' be an interpretation that has a witness J_w, and let M be an interpretation that extends M'. Intuitively, M and M' are the current interpretation and some previous

interpretation which was used in some previous search. Consider a set U of non-false literals in M. Then also all literals in U are non-false in M'. Consider a literal $l \in \mathbf{D}_{\text{lits}} \cap U$. If $M(d_{J_w}(l)) = \mathbf{f}$ (i.e., we have found a disjunct that has become false since the last search), then d_{J_w} does not support M and hence is not a witness for it. It is therefore possible that no witness for M exists, and that U is the set of literals that should be false in each fixpoint model of $\mathcal{D}$ extending M. As such, l is a good candidate cycle source (in M). If on the other hand $M(d_{J_w}(l)) \neq \mathbf{f}$, then J_w may still be a witness for M. Possibly J_w violates support of M in another literal $l' \in U$, but in that case l' will be a cycle source for the same reason. This inspires the following definition:

Definition 6.6.9 (Cycle sources) *Let M, M' be interpretations with M an extension of M', and let J_w be a witness for M'. Then the set CS of* cycle sources *(with respect to M, M' and J_w) is defined as $CS = \{l \in \mathbf{D}_{lits} \mid M(l) \neq \mathbf{f} \wedge M'(d_{J_w}(l)) \neq \mathbf{f} \wedge M(d_{J_w}(l)) = \mathbf{f}\}$.*

Let M, M', J_w and CS be as in the definition. Then there exists a justification J_s that supports M, such that $d_{J_s}(l) = d_{J_w}(l)$ for each $l \in \mathbf{D}_{\text{lits}} \setminus CS$. For this justification, the set $\{l \in \mathbf{D}_{\text{lits}} \mid d_{J_s}(l) \neq d_{J_w}(l)\}$ is equal to CS; for other justifications that support M, this set is a superset of CS.

Example 6.6.10 *Let $\mathcal{D} =$*

$$\left\lfloor \begin{array}{c} p \leftarrow q \vee a \vee b \\ \lceil\ q \leftarrow p\ \rceil \end{array} \right\rfloor,$$

and consider $M_1 = \emptyset$. The justification J_w with $d_{J_w}(p) = a$ is a witness for M_1. Consider $M_2 = M_1[b/\mathbf{f}]$; since a is still unknown, J_w is also a witness for M_2. There are therefore no cycle sources. Finally, consider $M_3 = M_2[a/\mathbf{f}] = \{a \mapsto \mathbf{f}, b \mapsto \mathbf{f}\}$. Then CS with respect to M_3, M_2 and J_w is the singleton $\{p\}$. Thus p may be contained in a set of literals which should be false in each fixpoint model of $\mathcal{D}$ extending M_3, which means that p may be false in each fixpoint model of $\mathcal{D}$ extending M_3. That is indeed the case.

Definition 6.6.11 (Witness up to CS) *A justification J is a witness for interpretation M* up to a set CS of literals*, if J supports M, and for any positive least loop in J, either all atoms in the positive loop are false in M, or the loop contains an atom of CS, and for any negative greatest loop in J, either all literals in the negative loop are false in M, or the negative loop contains a negative literal of CS.*

If J is a witness for M up to CS, and Algorithm 5 applied on $cs \in CS$ (and J) produces a local witness for M in cs, then afterwards, J is a witness for M up

to $CS \setminus \{cs\}$. Therefore, by repetitive application of Algorithm 5 on all literals of CS, either a set of defined literals that are false in each model of $\mathcal{D}$ extending M, or a witness for M up to the empty set, i.e., a witness for M, will be found.

The above reasoning requires that Algorithm 5 satisfies an additional invariant: namely that J is a witness up to CS, where J is the justification on which Algorithm 5 makes its local adjustments. It can be easily verified that this is indeed the case—the algorithm only changes $d_J(l)$ for literals $l \in HP$, in such a way that if any new positive least loops or any negative greatest loops are created, they certainly already contain a cycle source.

Algorithm 6 applies this repetitive procedure. It assumes J_w and J_s to be given, where J_w is a witness for the previous interpretation M', and J_s is a supporting justification for the current interpretation M. The application of Algorithm 5 may change J_s, as well as CS in the way indicated above.

Algorithm 6: FindWitness(M, J_s, J_w, $\mathcal{D}$)

Result: A set of literals that are false in each model of $\mathcal{D}$ extending M if it exists; else, the empty set, and J_s is witness for M.

```
1 Let CS := {l ∈ D_lits | d_{J_s}(l) ≠ d_{J_w}(l)} ;
2 while CS ≠ ∅ do
3 |   Choose a cs ∈ CS ;
4 |   Let U := the result of Algorithm 5 on cs ;
5 |   if U ≠ ∅ then return U ;
6 |   % Else: CS has become strictly smaller
7 return ∅ ;
```

Proposition 6.6.12 *Let M, J_w and J_s be as given above. The Algorithm 6, when called on CS, either returns a set of defined literals that are false in each model of $\mathcal{D}$ extending M, or returns the empty set. In the second case, the resulting justification J_s is a witness for M.*

Remark 6.6.13 *When Algorithm 6 returns the empty set and a witness, this witness can be used the next time cycle sources are needed. Using the most recent witness for this purpose has the advantage that then the produced set of cycle sources is small, since few literals will have become false in the mean time.*

Recall that we still require an *initial* search for a $\mathcal{D}$-witness. Actually, this initial search can be performed by Algorithm 6 except that the set CS of cycle sources in Algorithm 6 is changed to be the set consisting of all defined atoms a such that a is locally defined in a least fixpoint subdefinition of $\mathcal{D}$ and a is non-false in the current interpretation M, and all negative literals $\neg b$ such that b is locally

defined in a greatest fixpoint subdefinition of $\mathcal{D}$ and b is non-true in the current interpretation M.

Algorithm 7: FindInitialWitness(M, J_s, $\mathcal{D}$)

Result: A set of literals that are false in each model of $\mathcal{D}$ extending M if it exists; else, the empty set, and J_s is the initial witness for M.

Let $CS := \{a \mid a$ is locally defined in a least fixpoint subdefinition of $\mathcal{D}$ and $M(a) \neq \mathbf{f}\} \cup \{\neg b \mid b$ is locally defined in a greatest fixpoint subdefinition of $\mathcal{D}$ and $M(b) \neq \mathbf{t}\}$;
while $CS \neq \emptyset$ **do**
    Choose a $cs \in CS$;
    Let $U :=$ the result of Algorithm 5 on cs ;
    if $U \neq \emptyset$ **then return** U ;
    % Else: CS has become strictly smaller
return $\emptyset$;

6.7 MX(FO(FD))

We have implemented a prototype SAT(FD) solver on the basis of an existing open source SAT solver, namely MiniSat [60]. The resulting solver is called MiniSat(FD). MiniSat(FD) is used as a propositional model generator in a model expansion solver for FO(FD). In this section, we introduce the model expansion solver for FO(FD), called MX(FO(FD)).

6.7.1 Model expansion

Given a theory, *model expansion* is the computational task of finding a model that expands a given interpretation of a given subvocabulary of the theory. This task has many practical applications. Mitchell and Ternovska [130] proposed model expansion for FO and FO(ID) as a framework for solving NP problems.

Definition 6.7.1 (MX) *Let $\mathcal{L}$ be a logic.* Model expansion for $\mathcal{L}$, *denoted $\mathcal{L}$-MX is the following decision problem. An instance of the problem consists of a $\mathcal{L}$-theory T, a vocabulary $\sigma \subseteq vocab(T)$, where $vocab(T)$ is the vocabulary of T, and a finite σ-interpretation I_σ. The problem is to decide whether there exists a $vocab(T)$-interpretation I such that $I \models T$ and $I|_\sigma = I_\sigma$.*

We denote $\mathcal{L}$-MX simply by MX if $\mathcal{L}$ is clear from the context. The vocabulary σ is called the *input vocabulary*, the vocabulary $vocab(T) \setminus \sigma$ is called the *expansion vocabulary*, and the $vocab(T)$-model I is called an *expansion model of I_σ for T*.

Consider $\mathcal{L} = \text{FO}$ and $\mathcal{L} = \text{FO(ID)}$. It was proved that FO-$MX$ and FO(ID)-MX are NEXP-complete in [130].

Definition 6.7.2 (Parameterized MX) *Let $\mathcal{L}$ be a logic, T a $\mathcal{L}$-theory, σ a subvocabulary of $vocab(T)$. Then the problem $\mathcal{L}$-$MX_{\langle T,\sigma\rangle}$ is the problem of deciding, for a given finite σ-interpretation I_σ, whether there exists an expansion model of I_σ for T.*

Observe that if $\sigma = vocab(T)$, then $MX_{\langle T,\sigma\rangle}$ reduces to model checking, while if $\sigma = \emptyset$, the problem is that of deciding the existence of a model of T with a given finite size. Hence (parameterized) model expansion generalizes finite model generation for the case where the model size is given.

For a given $\mathcal{L}$-theory T, input vocabulary σ, and decision problem X on finite σ-interpretations, we say that $\mathcal{L}$-$MX_{\langle T,\sigma\rangle}$ *expresses* X when a finite σ-interpretation I_σ belongs to X iff there exists an expansion model of I_σ for T. It was proved in [130] that parameterized model expansion for FO and for FO(ID) *captures* NP. This means the following:

- for any T and $\sigma \subseteq vocab(T)$, the problem $MX_{\langle T,\sigma\rangle}$ is in NP;
- for any NP decision problem X on the class of finite σ-interpretations, there is a theory T with $vocab(T) \supseteq \sigma$ such that $MX_{\langle T,\sigma\rangle}$ expresses X.

In general, the expressivity of the logic $\mathcal{L}$ determines the complexity class that is captured by $\mathcal{L}$-MX and by $\mathcal{L}$-$MX_{\langle\cdot,\cdot\rangle}$. This means that parameterized MX is ideally suited as a computational paradigm for solving search problems (with logic). In this paradigm, a problem is solved by computing expansion models of a specific theory in some logic $\mathcal{L}$. To this end, a human expert should model the theory such that its expansion models corresponds to the solutions of the problem at hand.

6.7.2 The MX(FO(FD)) system

The previous section motivates the use of model expansion as a computational paradigm. Following this motivation, we have built an FO(FD)-MX solver called MX(FO(FD)).

The MX(FO(FD)) system consists of two components: a grounder, which was implemented by Johan Wittocx, and a propositional model generator, MiniSat(FD).

The grounder's task is to transform a given FO(FD)-MX problem with a given input interpretation to a PC(FD) theory. The output language of the grounder is Extended CNF (ECNF), a normal form for PC(FD) theories. The propositional solver for the MX(FO(FD)) system, MiniSat(FD), has been discussed at length in the previous sections of this chapter.

Both SAT(FD) and FO(FD)-MX solver are new research domains; no organized benchmarking suites exist for them yet. In the following, we describe a hand-crafted benchmark which is formalized in FO(FD) and report on MX(FO(FD))'s performance on this benchmark.

The platform on which these experiments were run is an Intel Core 2 Duo 3GHz with 2GB RAM running Kubuntu 7.10 linux.

A model checking example

MX(FO(FD)) is able to solve model checking problems by reading a given transition system and combining it with a formula specification that is to be verified in the transition system. We benchmark a simple fairness conditions checking problem. We did a preliminary experimental evaluation of MX(FO(FD) for model checking of fairness conditions.

We use the following μ-calculus expression for fairness conditions, which is presented in [107]:

$$\nu X.\mu Y.([-].(\langle a\rangle X) \vee Y).$$

This μ-calculus expression represents that a state in the transition system is fair if on all infinite paths, an a-labeled edge is infinitely often taken.

The above μ-calculus expression can be phrased in FO(FD), which is illustrated by the following FO(FD)theory:

$$\left[\begin{array}{l} \forall x\ (P(x) \leftarrow Q(x)) \\ \lfloor\ \forall x\ (Q(x) \leftarrow \forall y\ (Edge(x,y) \supset ((L(y,a) \wedge P(y)) \vee Q(y)))) \ \rfloor \end{array}\right],$$

where the relations P and Q express that a state in which an a-labeled edge is enabled infinitely often along all infinite paths, L is the labeling relation, expressing that a state has a certain label, and $Edge$ is the transition relation.

The experiments were done on the transition graph depicted in Figure 6.7. The results of these experiments are shown in Figure 6.8. The task consists of a model expansion problem, where the transition and labelling relationships are known, to decide which nodes are fair. Grounding times are included in the results.

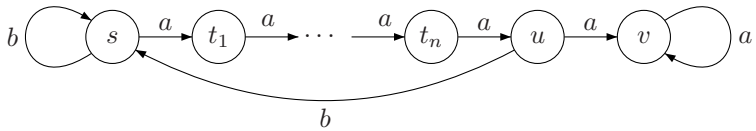

Figure 6.7: A transition graph

$\|nodes\|$	time(sec)
503	0.009
1503	0.18
2503	18.79

Figure 6.8: Model checking of fairness conditions

From these experimental results, we can conclude that fairness conditions can be evaluated efficiently using MX(FO(FD)). In [96], the same experiment was made using a reduction from μ-calculus to ASP, where similar results were obtained (higher execution times, but on a slower machine).

6.7.3 Applications of FO(FD) and MX(FO(FD))

Many applications can be found on the use of fixpoint expressions, mainly for inductive definitions and for coinductive definitions, which have alternating depth of one, e.g., modelling properties like transitive closure (reachability), bisimulation, temporal expressions from the logics LTL and CTL, etc.

There are many interesting applications for nested induction and coinduction expressions, which we will address in more detail.

An important application domain for fixpoint definitions can be found in verification of automata. Temporal logics like CTL, LTL and CTL* allow to express time-variant properties of automata such as termination, starvation/liveness and fairness. The μ-calculus is a superset logic of those temporal logics and is bound on fixpoint expressions. As μ-calculus can be transformed into fixpoint definitions, any applications of model checking and model generation of temporal logics can also be described in FO(FD). For example the notion of fairness, namely, some condition is infinitely often true, can only be expressed by nested fixpoint expressions. An example of this was worked out in the previous section.

Another application domain is so-called *parity games*, which are infinite games played on a graph in which the nodes are annotated with priorities. For more information we refer to [66]. The winning strategies in a parity game can be

expressed in FO(FD) using a transformation in which the nesting depth increases polynomially with the number of priorities.

6.8 Conclusions and related work

In this chapter, we presented a standard algorithm for the evaluation of fixpoint expressions that was proposed in [27]. We have developed an algorithm for the computation of fixpoint definitions based on Browne et al.'s algorithm. We have given a complexity result for SAT(FD). We introduced justifications for propositional fixpoint definitions and defined several related concepts that were useful for constructing SAT(FD) algorithms.

We have discussed how SAT(FD) solvers can be built by extending SAT solvers with an additional propagation mechanism suitable for reasoning on fixpoint definitions. Also, we have developed algorithms to solve the SAT(FD) problem. We have implemented the algorithms as a prototype SAT(FD) solver, MiniSat(FD). We also presented a model expansion solver for FO(FD): MX(FO(FD)). Such a model expansion system is composed of a grounder and MiniSat(FD). We evaluated the performance of MX(FO(FD)) by an example from the model checking domain.

In the context of fixpoint logics, according to our knowledge, SAT(FD) is the first work to extend SAT solvers with techniques to handle fixpoint expressions. In the context of FO(ID), there are several SAT(ID) solvers, e.g., [117, 118], and the IDP system [116, 182] is the model expansion solver for FO(ID). In the context of ASP, many ASP solvers have been developed and implemented. We mention some: cmodels [105], SModels [167], DLV [47, 103], clasp [68].

There are several other solvers for solving the satisfiability and validity problems for modal fixpoint logics, e.g., [66, 65]. The main difference with our work is that our computation has to be integrated with SAT technology. We focus on the detection of witnesses, and handle the rest of the computation with unit propagation also used in SAT. In contrast, the underlying technique of referenced works is based on characterizations of satisfiability through infinite (cyclic) tableaux in which branches have an inner thread structure mirroring the regeneration of least and greatest fixpoint constructs in the infinite. Well-foundedness for unfoldings of least fixpoints is checked using deterministic parity automata.

Chapter 7

Conclusion

The first part of this text studied the deductive inference methods for FO(ID). The second part of this text studied the extension of first-order logic with nested induction and coinduction. We summarize the contributions of this thesis and provide some directions for future work.

7.1 Summary of contributions

In the first part of this text, we have studied a more traditional form of inference: deduction, for FO(ID). By developing formal proof systems for FO(ID), we want to enhance the understanding of proof-theoretic foundations of FO(ID).

We have developed a formal proof system for PC(ID) based on the sequent calculus (Gentzen-style deductive system). We have defined a new logic FO(ID,SLFP), which is an integration of FO(ID) with stratified least fixpoint logic. We have developed a sequent calculus proof system for FO(ID, SLFP). By restricting input to FO(ID), this proof calculus yields a proof system for FO(ID) then. However, in proofs of FO(ID) formulas, some inference rules may introduce least fixpoint expressions, and thus, proofs may still contain expressions of FO(ID,SLFP).

In the second part of this text, we have studied a new logic FO(FD). We have made a study on the relationship between FO(FD) and FO(ID). We have made a comparison of FO(FD) and coinductive logic programming.

We have made an algorithm study of the computation of fixpoint expressions. We have made a semantical study of PC(FD), giving an alternative characteriza-

tion of the models of propositional fixpoint definitions. We have made an algorithm study of SAT(FD), providing several algorithms to solve the SAT(FD) problem. Also, we have implemented the MiniSat(FD) system, a prototype SAT(FD) solver, as an extension of an existing SAT solver.

Finally, we have studied methodological principles of modelling in FO(FD), presented the MX(FO(FD)) system for model expansion for FO(FD), and demonstrated its performance.

7.2 Future work

The first part of the work could still be extended in several ways. One application of this work could be for the development of tools to check the correctness of the outputs generated by FO(ID) model generators such as the IDP system. Given a FO(ID) theory T as input, such a solver outputs a model for T or concludes that T is unsatisfiable. In the former case, an independent *model checker* can be used to check whether the output is indeed a model of T. However, when the solver concludes that T is unsatisfiable, it is less obvious how to check the correctness of this answer. One solution is to transform a trace of the solvers computation into a proof of unsatisfiability in a FO(ID) proof system. An independent *proof checker* can then be used to check this formal proof. Model and proof checkers can be a great help to detect bugs in model generators. An analogous checker for the Boolean Satisfiability problem (SAT) solvers was described in [184].

On the longer run, we view our work as a first step towards the development of decidable fragments of FO(ID). A potential use of this is in the field of description logics. Deductive reasoning is the distinguished form of inference of Description Logics. Given the efforts to extend Description Logics with rules and the fact that FO(ID) offers a natural, clean integration of a very useful form of rules in FO, it seems that research on decidable fragments of FO(ID) could play a useful role in that area.

There are several ways in which the second part of the work could be extended.

First, since coinductive logic programming is strongly related to fixpoint definitions in FO(FD), a formal embedding of coinductive logic programming in FO(FD) can be defined; the benefits of such an embedding should be demonstrated.

Second, a number of SAT solvers have been developed for difference logic [137], in particular in the context of the satisfiability modulo theories (SMT). We can provide translations from PC(FD) theories to theories of difference logic. The translations reduce SAT(FD) to finding satisfying interpretations of difference logic theories. Hence, SAT solvers for difference logic can be used for SAT(FD) without

any modifications. This provides a novel way of implementing SAT(FD) solvers where it is possible to exploit directly the rapidly improving SMT solver technology. Related work can be found in [136, 90], where the translations from logic programs to difference logic theories are developed and the potential of SMT solvers in the computation of stable models using these translations is evaluated. In fact, such a study is being conducted in our research group.

Finally, a complexity analysis of the algorithms for SAT(FD) should be provided. It would be interesting to see how adequate the algorithms are with respect to the inherent complexity of the implemented tasks.

Appendix A

Extended Clausal Normal Form (ECNF)

In this appendix, we define the ASCII syntax for ECNF theories. For the formal definition of the logical syntax of ECNF theories, we refer to Section 6.2.2.

The syntax is derived from the DIMACS syntax for CNF theories. In this syntax, the following conventions are used:

- atoms are represented by strictly positive integers;
- the negation of an atom N is represented by $-N$;
- clauses are represented by whitespace-separated lists of literals, and are terminated by a 0.

The syntax further allows *comments*: lines that start (first symbol on a new line) by a "c". It further contains a notation for a *problem statement*: a line that starts by "p cnf", and further contains two positive integers, representing respectively the largest atom that occurs (positively or negatively) in the theory, and the number of clauses in the theory. This problem statement should occur before any clauses (but potentially after some comments). In practice, most contemporary SAT solvers disregard this information, which is furthermore difficult to produce for a grounder.

Example A.0.1 *The CNF theory*

$$x_1 \vee \neg x_2 \vee x_3,$$
$$\neg x_1 \vee x_2,$$
$$\neg x_3 \vee \neg x_1,$$
$$x_1 \vee x_2 \vee \neg x_3,$$

can be represented in DIMACS CNF as follows:

```
c This is a comment.
p cnf 3 4
1 -2 3 0
-1 2 0
-3 -1 0
1 2 -3 0
```

Apart from the comment and problem statement line, a DIMACS CNF file contains only integers and whitespace. In the following, we will implicitly assume any list and any separately mentioned item to be whitespace-separated, and literals to be represented as in DIMACS CNF.

We begin by modifying the problem statement: the problem statement of an ECNF theory starts by "p ecnf", and is followed by a (whitespace-separated) list of extensions that are used in the theory.

- if inductive definitions occur, "def" must be in the list;
- if fixpoint definitions occur, "fdef" must be in the list.

A.1 Inductive definitions

An ECNF theory may contain one inductive definition that is in DefNF normal form. Recall that a DefNF definition contains exactly one rule for each defined atom, and that each rule body is either a disjunction or a conjunction of literals.

We represent a disjunctively defined rule by "D", and a conjunctively defined rule by "C". This declaration is then followed first by the atom being defined, next by a list of body literals, and finally terminated by a 0. Note in particular that both the empty disjunctive body (false) and the empty conjunctive body (true) can be represented, by following the atom being defined by an empty list of body literals.

Example A.1.1 *We represent the ECNF theory*

$$\left\{ \begin{array}{c} x_1 \vee \neg x_2 \vee x_3, \\ x_1 \leftarrow \neg x_2 \vee x_3 \\ x_2 \leftarrow x_1 \wedge x_3 \end{array} \right\}$$

in this format:

```
p ecnf def
1 −2 3 0
D 1 −2 3 0
C 2 1 3 0
```

There is no requirement that a set of rules be represented contiguously; other expressions, such as clauses, may be interspersed.

A.2 Fixpoint definitions

An ECNF theory may contain one fixpoint definition that is in DefNF normal form. Since such a (root) fixpoint definition has subdefinitions, it is required to encode multiple fixpoint definitions. Root and child fixpoint definitions are encoded by numbers. The root fixpoint definition is encoded by 1, and if fixpoint definition $\mathcal{D}_1$ is encoded by n_1, fixpoint definition $\mathcal{D}_2$ is encoded by n_2 and $\mathcal{D}_2$ is a subdefinition (child) of $\mathcal{D}_1$, then $\mathsf{n}_1 < \mathsf{n}_2$.

We first represent the structure of a fixpoint definition. A fixpoint defiition is declared by "L", respectively by "G", if it is a least fixpoint definition, respectively, a greatest fixpoint definition, followed by a positive integer that uniquely encodes the fixpoint definition, followed by a list of positive integers that encodes its children (subdefinitions), terminated by 0.

Then the rules in the fixpoint definition are represented as follows. A disjunctively defined rule, respectively a conjunctively defined rule, in a fixpoint definition is declared by "FP D", respectively by "FP C". This declaration is then followed first by a positive integer that encodes the fixpoint definition to which the rule belongs, next by a list of body literals, and finally, terminated by 0.

Example A.2.1 *We represent the ECNF theory*

$$\left\lfloor \begin{array}{c} x_1 \vee \neg x_2 \vee x_3, \\ x_1 \leftarrow x_2 \vee x_3 \\ \lceil\ x_2 \leftarrow x_1 \wedge x_3\ \rceil \end{array} \right\rfloor$$

in this format:

```
p ecnf fdef
1 −2 3 0
L 1 2 0
G 2 0
FP D 1 1 2 3 0
FP C 2 2 1 3 0
```

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addisin-Wesley Publishing Company, 1995.

[2] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland Publishing Company, 1977.

[3] L. Afanasiev, T. Grust, M. Marx, J. Rittinger, and J. Teubner. An Inflationary Fixed Point Operator in XQuery. In *ICDE 2008*, pages 1504–1506.

[4] H. R. Andersen. Model checking and boolean graphs. In B. Krieg-Bruckner, editor, *Proc. 4th European Symp. on Programming*, volume 582 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 1992.

[5] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[6] K. R. Apt and M.H. Van Emden. Contributions to the Theory of Declarative Knowledge. *Journal of the ACM*, 29(3):841–862, 1982.

[7] D. N. Arden. Delayed logic and finite state machines. In *Theory of Computing Machine Design*, pages 1–35. Univ. of. Michigan Press, 1960.

[8] A. Arnold and P. Crubille. A linear algorithm to solve fix-point equations on transition systems. *IPL*, 29:57–66, 1988.

[9] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, 2002.

[10] J. W. Bakker. *Recursive Procedures*. Mathematical Centre Tracts 24, Mathematisch Centrum, Amsterdam, 1971.

[11] J. W. Bakker. The fixed-point approach in semantics: theory and applications. In J. W. Bakker, editor, *Foundations of Computer Science*, pages 3–53. North Holland, 1975.

[12] J. W. Bakker and W. P. Roever. A calculus for recursive program schemes. In M. Nivat, editor, *Proc. IRIA symposium on on Automata, Languages and Programming*, pages 167–196. North Holland, 1973.

[13] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[14] C. Baral, G. Brewka, and J. S. Schlipf, editors. *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*. Springer, 2007.

[15] J. Barwise and J. Etchemendy. *The Liar: an Essay in Truth and Circularity*. Oxford University Press, 1987.

[16] J. Barwise and L. Moss. *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*. CSLI Publications, 1996.

[17] H. Bekič. *Definable operations in general algebras and the theory of automata and flowcharts*. 1969. Unpublished Manuscript.

[18] N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37, 1977.

[19] N. Bidoit and M. Ykhlef. Fixpoint Calculus for Querying Semistructured Data. In *WebDB'98*, volume 1590 of *Lecture notes in Computer Science*, pages 78–97. Springer, 1999.

[20] A. Blass and Y. Gurevich. Existential fixed-point logic. In E. Börger, editor, *Computation Theory and Logic*, volume 270 of *Lecture notes in Computer Science*, pages 20–36. Springer-Verlag, 1987.

[21] A. Blikle. A comparative review of some program verification methods. In J. Gruska, editor, *6th Symposium on Mathematical Foundations of Computer Science (MFCS'77)*, volume 53 of *Lecture Notes in Computer Science*, pages 17–33. Springer, 1977.

[22] G. Boolos. Don't eliminate cut. *Journal of Philosophical Logic*, 13(4):373–378, 1984.

[23] R. J. Brachman and H. J. Levesque. Competence in knowledge representation. In *National Conference on Artificial Intelligence (AAAI'82)*, pages 189–192, 1982.

[24] J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, 1998.

[25] M. Brandt and F. Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticae*, 33(4):309–338, 1998.

[26] J. Brotherston. *Sequent Calculus Proof System for Inductive Definitions*. PhD thesis, University of Edinburgh, 2006.

[27] A. Browne, E.M. Clarke, S. Jha, D.E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178:237–255, 1997.

[28] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput*, C-35(8):677–691, 1986.

[29] W. Buchholz, S. Feferman, and W. Pohlers W. Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*, volume 897 of *Lecture Notes in Mathematics*. Springer-Verlag.

[30] H. K. Büning and X. Zhao, editors. *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2008*, volume 4996 of *Lecture Notes in Computer Science*. Springer, 2008.

[31] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Annual Symp on Logic in Computer Science*, pages 428–439. IEEE Computer Soc. Press, 1990.

[32] S. R. Buss. *Handbook of Proof Theory*. Elsevier, 1998.

[33] D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive Description Logics with fixpoints based on automata on infinite trees. In *IJCAI'99*, pages 84–89.

[34] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[35] E. M. Clarke. Program invariants as fixed points (preliminary reports). In *FOCS*, pages 18–29. IEEE, 1977.

[36] E. M. Clarke, I. A. Draghicescu, and R. P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. In A. Arnold and N. D. Jones, editors, *Proc. 15th Coll. on Trees in Algebra and Programming*, volume 507 of *Lecture Notes in Computer Science*. Springer, 1990.

[37] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, editor, *Logic of Programs: Workshop*, volume 131 of *Lecture Notes in Computer Science*. Springer, 1981.

[38] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages Systems*, 8(2):244–264, 1986.

[39] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Inform*, 27(8):725–747, 1990.

[40] K.J. Compton. Stratified least fixpoint logic. *Theoretical Computer Science*, 131(1):95–120, 1994.

[41] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.

[42] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Proc. 1989 Internat. Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373. Springer, 1989.

[43] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[44] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[45] J. W. de Bakker. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.

[46] W. P. de Roever. On backtracking and greatest fixpoints. In A. Salomaa and M. Steinby, editors, *Fourth Colloquium on Automata, Languages and Programming (ICALP)*, volume 52 of *Lecture Notes in Computer Science*, pages 412–429, 1977.

[47] T. Dell'Armi, W. Faber, G. Ielpa, C. Koch, N. Leone, S. Perri, and G. Pfeifer. System description: DLV. In T. Eiter, W. Faber, and M. Truszczyński, editors, *LPNMR*, volume 2173 of *Lecture Notes in Computer Science*, pages 424–428. Springer, 2001.

[48] M. Denecker. *Knowledge Representation and Reasoning in Incomplete Logic Programming*. PhD thesis, Department of Computer Science, K.U.Leuven, 1993.

[49] M. Denecker. The well-founded semantics is the principle of inductive definition. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence (JELIA'98)*, volume 1489 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer-Verlag, 1998.

[50] M. Denecker. Extending classical logic with inductive definitions. In J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *CL*, volume 1861 of *Lecture Notes in Computer Science*, pages 703–717. Springer, 2000.

[51] M. Denecker, M. Bruynooghe, and V. Marek. Logic programming revisited: Logic programs as inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, 2(4):623–654, October 2001.

[52] M. Denecker and D. De Schreye. Justification semantics: A unifiying framework for the semantics of logic programs. In Luís Moniz Pereira and Anil Nerode, editors, *LPNMR*, pages 365–379. MIT Press, 1993.

[53] M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In V. Lifschitz and I. Niemelä, editors, *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2004.

[54] M. Denecker and E. Ternovska. Inductive situation calculus. *Artificial Intelligence*, 171(5-6):332–360, April 2007.

[55] M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions. *ACM Transactions On Computational Logic (TOCL)*, 9(2), 2008.

[56] M. Denecker and J. Vennekens. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In Baral et al. [14], pages 84–96.

[57] M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczyński. The second answer set programming competition. In *LPNMR09*, pages 637–654, 2009.

[58] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[59] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, second edition, 1999.

[60] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[61] E. A. Emerson. Real-time and the Mu-Calculus (preliminary report). In *REX Workshop Mook'91*, volume 600 of *Lecture notes in Computer Science*, pages 176–194. Springer, 1992.

[62] E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. 1st Annual Symp. on Logic in Computer Science*, pages 267–278. IEEE Computer Soc. Press, 1986.

[63] M. J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. System Sci*, 18:194–211, 1979.

[64] R. W. Floyd. Assigning meaning to programs. In *Proc. Symposia in Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, 1967.

[65] O. Friedmann and M. Lange. A solver for modal fixpoint logics. In *Proceedings of the 6th Workshop on Methods for Modalities, M4M-6*, 2009.

[66] O. Friedmann and M. Lange. Solving parity games in practice. In *Proc. 7th Int. Symp. on Automated Technology for Verification and Analysis, ATVA'09*, volume 5799 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2009.

[67] V. Gapeyev, M. Y. Levin, and B. C. Pierce. Recursive subtyping revealed. *Journal of Functional Programming*, 12(6):511–548, 2002.

[68] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *clasp*: A conflict-driven answer set solver. In Baral et al. [14], pages 260–265.

[69] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczyński. The first answer set programming system competition. In Baral et al. [14], pages 3–17.

[70] M. Gebser and T. Schaub. Tableau calculi for answer set programming. In S. Etalle and M. Truszczynski, editors, *Logic Programming, 22nd International Conference, ICLP 2006*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer, 2006.

[71] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47(1):185–221, 1993.

[72] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[73] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *International Joint Conference and Symposium on Logic Programming (JICSLP'88)*, pages 1070–1080. MIT Press, 1988.

[74] G. Gentzen. Untersuchungen über das logische schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[75] V. Gimona Giarratana and U. Montanari. Observability concepts in abstract data type specification. In A. Mazurkievicza, editor, *5th Symposium on Mathematical Foundations of Computer Science*, volume 45 of *Lecture Notes in Computer Science*, pages 576–587. Springer, 1976.

[76] S. Ginsburg and H. G. Rice. Two families of languages related to algol. *J. ACM*, 9(3):350–371, 1962.

[77] J. Goguen and K. Lin. Behavioral verification of distributed concurrent systems with bobj. In *Proc. Conference on Quality Software*, pages 216–235. IEEE Press, 2003.

[78] D. Goldin and D. Keil. Interaction, evolution and intelligence. In *Proc. Congress on Evolutionary Computing*, pages 805–814, 2001.

[79] A. Gordon. A tutorial on co-induction and functional programming. In *Proceedings of the 1994 Glasgow Workshop on Functional Programming, Springer Workshops in Computing (Functional Programming)*, pages 78–95, 1995.

[80] E. Grädel and S. Kreutzer. Will deflation lead to depletion? On non-monotone fixed point inductions. In *IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 158–167, 2003.

[81] G. Gupta, A. Bansal, R. Min, L. Simon, and A. Mallya. Coinductive Logic Programming and Its Applications. In V. Dahl and I. Niemelä, editors, *ICLP*, volume 4670 of *Lecture notes in Computer Science*, pages 27–44. Springer, 2007.

[82] Y. Gurevich and S. Shelah. Fixed-point Extensions of First-Order Logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[83] M. Hagiya and T. Sakurai. Foundation of Logic Programming Based on Inductive Definition. *New Generation Computing*, 2:59–77, 1984.

[84] P. Hou and M. Denecker. A deductive system for FO(ID) based on least fixpoint logic. In E. Erdem, F. Lin, and T. Schaub, editors, *LPNMR*, volume 5753 of *Lecture notes in Computer Science*, pages 129–141. Springer, 2009.

[85] P. Hou and M. Denecker. A logic of fixpoint definitions. In *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP) 2009*, pages 61–75, 2009.

[86] P. Hou, J. Wittocx, and M. Denecker. A deductive system for PC(ID). In Baral et al. [14], pages 162–174.

[87] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[88] B. Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Draft manuscript.

[89] J. Jaffar and P. J. Stuckey. Semantics of infinite tree LP. *TCS*, 46(2-3):141–158, 1986.

[90] T. Janhunen, I. Niemelä, and M. Sevalnev. Computing stable models via reductions to difference logic. In E. Erdem, F. Lin, and T. Schaub, editors, *LPNMR*, volume 5753 of *Lecture notes in Computer Science*, pages 142–154. Springer, 2009.

[91] M. Järvisalo, T. A. Junttila, and I. Niemelä. Justification-based non-clausal local serach for SAT. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, editors, *ECAI*, pages 535–539. IOS Press, 2008.

[92] K. Gödel. Ueber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatsh. Math. Phys.*, 37:349–360, 1931.

[93] G. Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475. North-Holland, 1974.

[94] A. C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[95] S. Kanger. A simplified proof method for elementary logic. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 87–94, 1963.

[96] M. Keinänen and I. Niemelä. Solving alternating boolean equation systems in answer set programming. In D. Seipel, M. Hanus, U. Geske, and O. Bartenstein, editors, *INAP/WLP*, volume 3392 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2004.

[97] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[98] G. Kreisel. Generalized inductive definitions. Technical report, Section III in the Stanford University report on the Foundations of Analysis, 1963.

[99] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[100] P. J. Landin. Correspondence between algol 60 and church's lambda-notation: Part I. *Commun. ACM*, 8(2):89–101, 1965.

[101] P. J. Landin. Correspondence between algol 60 and church's lambda-notation: Part II. *Commun. ACM*, 8(3):158–167, 1965.

[102] K. G. Larsen. Efficient local correctness checking. In G. V. Bochmann and D. K. Probst, editors, *Proc. 4th Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1992.

[103] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *CoRR*, cs.AI/0211004, 2002.

[104] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[105] Y. Lierler. cmodels - SAT-based disjunctive answer set solver. In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, pages 447–451. Springer, 2005.

[106] F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by sat solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.

[107] X. Liu, C. R. Ramakrishnan, and S. A. Smolka. Fully local and efficient evaluation of alternating fixed points (extended abstract). In B. Steffen, editor, *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 1998.

[108] A. Livchak. The Relational Model for Process Control. *Automatic Documentation and Mathematical Linguistics*, 4:27–29, 1983.

[109] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[110] J. Loeckx and K. Sieber. *Foundations of Program Verification*. Wiley, 1984.

[111] A. Mader. Tableau recycling. In G. V. Bochmann and D. K. Probst, editors, *Proc. 4th Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*. Springer, 1992.

[112] A. Mallya. *Multivalued Deductive Multi-valued Model Checking*. PhD thesis, UT Dallas, 2006.

[113] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

[114] M. Mariën, D. Gilis, and M. Denecker. On the relation between ID-Logic and Answer Set Programming. In J. J. Alferes and J. A. Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2004.

[115] M. Mariën, R. Mitra, M. Denecker, and M. Bruynooghe. Satisfiability checking for PC(ID). In Sutcliffe and Voronkov [171], pages 565–579.

[116] M. Mariën, J. Wittocx, and M. Denecker. The IDP framework for declarative problem solving. In *Search and Logic: Answer Set Programming and SAT*, pages 19–34, 2006.

[117] M. Mariën, J. Wittocx, and M. Denecker. Integrating inductive definitions in SAT. In Nachum Derschowitz and Andrei Voronkov, editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 378–392. Springer, 2007.

[118] M. Mariën, J. Wittocx, M. Denecker, and M. Bruynooghe. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2008*, Lecture Notes in Computer Science, pages 211–224. Springer, 2008.

[119] P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. E. Fenstad, editor, *Second Scandinavian Logic Symposium*, pages 179–216, 1971.

[120] A. W. Mazurkiewicz. Proving algorithms by tail functions. *Information and Control*, 18(3):220–226, 1971.

[121] A. W. Mazurkiewicz. Proving properties of processes. *Algorytmy*, 11:5–22, 1974.

[122] J. McCarthy. A basis for a mathematical theory of computation. In *Proc. Western Joint Computer Conf.*, volume 19, pages 225–238. Spartan Books, 1961.

[123] J. McCarthy. Towards a mathematical science of computation. In *Proceedings of IFIP Congress 62*, pages 21–28. North-Holland, 1963.

[124] J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the Fifth International Conference on Artificial Intelligence (IJCAI77)*, pages 1038–1044, 1977.

[125] J. McCarthy. Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artifical Intelligence*, 28:89–116, 1986.

[126] K. L. McMillan. *Symbolic model checking: an approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, 1992.

[127] R. Milner. A calculus of communicating systems. volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[128] R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87:209–220, 1991.

[129] D. G. Mitchell. A SAT solver primer. *Bulletin of the Europen Association for Theoretical Computer Science*, 85:112–132, 2005.

[130] D. G. Mitchell and E. Ternovska. A framework for representing and solving NP search problems. pages 430–435. AAAI Press/MIT Press, 2005.

[131] J. H. Morris. *Lambda-calculus models of programming languages*. PhD thesis, M.I.T, 1968.

[132] Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North-Holland Publishing Company, Amsterdam- New York, 1974.

[133] Y. N. Moschovakis. On non-monotone inductive definability. *Fundamenta Mathematicae*, 82(39-83), 1974.

[134] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC'01*, pages 530–535. ACM, 2001.

[135] M. A. Nait-Abdallah. On the interpretation of infinite computations in Logic Programming. In *Proc. ICALP 1984*, pages 358–370, 1984.

[136] I. Niemelä. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence*, 53:313–329, 2008.

[137] R. Nieuwenhuis and A. Oliveras. DPLL(t) with exhaustive theory propagation and its application to difference logic. In K. Etessami and S.K. Rajamani, editors, *CAV*, volume 3576 of *Lecture notes in Computer Science*, pages 321–334. Springer, 2005.

[138] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t). *J. ACM*, 53(6):937–977, 2006.

[139] D. Park. Fixpoint induction and proofs of program properties. *Machine Intelligence*, 5:59–78, 1970.

[140] D. Park. The Y-combinator in Scott"s lambda-calculus models. In *Symposium on Theory of Programming*, 1970. unpublished.

[141] D. Park. Finiteness is mu-ineffable. Technical Report 3, University of Warwick, 1974.

[142] D. Park. Finiteness is mu-ineffable. *Theoretical Computer Science*, 3:173–181, 1976.

[143] D. Park. On the semantics of fair parallelism. In *Proc. Abstract Software Specifications, Copenhagen Winter School*, Lecture Notes in Computer Science, pages 504–526, 1979.

[144] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science: 5th GI-Conference, Karlsruhe*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[145] B. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[146] E. Pontelli and T. Son. Justifications for logic programs under answer set semantics. In S. Etalle and M. Truszczyński, editors, *ICLP 2006*, volume 4079 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2006.

[147] E. Post. Formal reduction of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.

[148] R. Reiter. On Closed World Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

[149] R. Reiter. Equality and domain closure in first-order databases. *Journal of the ACM*, 27:235–249, 1980.

[150] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages, Book resulting from the Intervale Workshop*, pages 191–233, 1982.

[151] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, 1967.

[152] A. Roychoudhury, C. R. Ramakrishnan, and I. V. Ramakrishnan. Justifying proofs using memo tables. In *PPDP 2000*, pages 178–189. ACM Press, 2000.

[153] D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(4), 2009.

[154] J. S. Schlipf. Complexity and undecidability results in logic programming. *Annals of Mathematics and Artificial Intelligence*, 15:257–288, 1995.

[155] J. S. Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51:64–86, 1995.

[156] D. Scott. *A construction of a model for the λ-calculus*. Oxford, 1969. Manuscript.

[157] D. Scott. *Models for the λ-calculus*. Oxford, 1969. Manuscript.

[158] D. Scott. *A type-theoretical alternative to CUCH, ISWIM, OWHY*. Oxford, 1969. Typed script.

[159] D. Scott. Continuous lattices. In E. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer, 1972.

[160] D. Scott. The lattice of flow diagrams. In E. Engeler, editor, *Symposium of Semantics of Algorithmic Languages*, volume 188 of *Lecture Notes in Mathematics*, pages 311–366. Springer, 1972.

[161] D. Scott. Data types as lattices. *SIAM J. on Computing*, 5:522–587, 1976.

[162] D. Scott and J. de Bakker. *A theory of programs*. 1969. Handwritten notes.

[163] J. P. Marques Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[164] L. Simon. *Extending Logic Programming with Coinduction*. PhD thesis, UT Dallas, 2006.

[165] L. Simon, A. Mallya, A. Bansal, and G. Gupta. Coinductive logic programming. In S. Etalle and M. Truszczyński, editors, *ICLP 2006*, volume 4079 of *Lecture Notes in Computer Science*, pages 330–344. Springer, 2006.

[166] P. Simons. *Extending and Implementing the Stable Model Semantics*. PhD thesis, Helsinki University of Technology, 2000.

[167] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, 2002.

[168] C. Spector. Inductively defined sets of natural numbers. In *Infinitistic Methods (Proc. 1959 Symposium on Foundation of Mathematis in Warsaw)*, pages 97–102. Pergamon Press, Oxford, 1961.

[169] C. Stirling and D. J. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[170] R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81:249–264, 1989.

[171] G. Sutcliffe and A. Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*. Springer, 2005.

[172] M. E. Szabo, editor. *The Collected Papers of Gerhard Gentzen*. North-Holland Publishing Co., Amsterdam, 1969.

[173] G. Takeuti. *Proof Theory*. North-Holland Publishing Co., Amsterdam, 1975.

[174] A. Tarski. Lattice-theoretic fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5:285–309, 1955.

[175] G. S. Tseitin. On the complexity of derivation in the propositional calculus, *Zapiski nauchnykh seminarov. LOMI*, 8:234–259, 1968. English translation of this volume: Studies in Constructive Mathematics and Mathematical Logic, Part 2, A. O. Slisenko, eds. Consultants Bureau, N.Y., 1970, pp. 115-125.

[176] M. Vardi. The complexity of relational query languages. In *14th ACM Symposium on Theory of Computing*, pages 137–145, 1982.

[177] J. Vennekens and M. Denecker. FO(ID) as an extension of DL with rules. In *The 6th Annual European Semantic Web Conference*, pages 384–398, 2009.

[178] S. Wainer and L. Wallen. Basic proof theory. In H. Simmons P. Aczel and S. Wainer, editors, *Proof Theory*, pages 1–26. Cambridge University Press, 1992.

[179] H. Wang. Towards mechanical mathematics. *IBM Journal of Research and Development*, 4:2–22, 1960.

[180] P. Wegner and D. Goldin. Mathematical models of interactive computing. Technical Report CS 99-13, Brown University, 1999.

[181] G. Winskel. Model checking in the modal nu-calculus. In *Proc. 16th Internat. Coll. on Automata, Languages, and Programming*, 1989.

[182] J. Wittocx, M. Mariën, and M. Denecker. The IDP system: a model expansion system for an extension of classical logic. In *Search and Logic: Answer Set Programming and SAT*, pages 572–577, 2008.

[183] J. Wittocx, J. Vennekens, M. Mariën, M. Denecker, and M. Bruynooghe. Predicate introduction under stable and well-founded semantics. In S. Etalle and M. Truszczyński, editors, *ICLP*, volume 4079 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2006.

[184] L. Zhang and S. Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10880–10885. IEEE Computer Society, 2003.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Informatics Section
Celestijnenlaan 200A – B-3001 Heverlee, Belgium

KATHOLIEKE UNIVERSITEIT LEUVEN

ASSOCIATIE K.U.LEUVEN