KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Standard, Interactive and Lifted Model Expansions using FO(·)

Pierre Carbonnelle

Supervisors:
Prof. dr. Marc Denecker
Prof. dr. ir. Gerda Janssens

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

February 2024

Standard, Interactive and Lifted Model Expansions using FO(·)

Pierre CARBONNELLE

Examination committee:
Prof. dr. ir. Emmanuel Van Lil, chair
Prof. dr. Marc Denecker, supervisor
Prof. dr. ir. Gerda Janssens, supervisor
Prof. dr. ir. Tom Holvoet
Prof. dr. ir. Hendrik Blockeel
Prof. dr. ir. Annie Liu
(Stony Brook University)
Prof. dr. Alexander Felfernig
(Graz University of Technology)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

February 2024

© 2024 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Pierre Carbonnelle, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, 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, electronic or any other means without written permission from the publisher.

Preface

Unlike walking on the moon, completing a PhD is a small step for mankind, but a big step for a man like me. I remember being asked by my grand aunt when I was 6 years old: "What would you like to do when you grow older?" I said that I wanted to become a "savant" (in French), i.e., a scientist. Her amusement embarrassed me. For various reasons, I had a career in industry instead, but I recently went back to pursuing my original dream. Am I a scientist now? You are the judge, but at least, I can cross the line "PhD" on my bucket list.

Like walking on the moon, completing a PhD is only possible thanks to the help of many persons. My promotor, Prof. Marc Denecker, played an essential role. He is a true pioneer: he is exploring an uncharted field of research that is neglected by his peers. I do believe that his work is significant, and it was an honor to be accepted in his research team. I have tried to contribute to it the best I could.

I have not always been a docile PhD student: I lost patience when I had the impression that we were splitting hairs, and I grew frustrated when our first paper was rejected four times. At times, I doggedly followed my own path. To my defense, I have been trained as an engineer, not as a mathematician. I'm grateful that Marc has been very, very patient with me, and taught me the need to be precise in scientific communication, and, more generally, when communicating with others.

After I invented a faster method to solve generative configuration problems (Chapter 4 on page 65), I found it very difficult to explain it to others. Prof. Maurice Bruynooghe was instrumental in bringing clarity to my presentation, and in generalizing the method: I'm deeply indebted to him for his contributions.

My other colleagues also played an important role, each in different ways: Prof. Gerda Janssens, Prof. Vennekens, Prof. Bart Bogaerts, Jo Devriendt, Ingmar Dasseville, Simon Vandevelde, Simon Marynissen, Djordje Markovic and Linde Vanbesien. It would be too long to describe their contributions here, but I

heartily thank them.

I'm also indebted to my beloved wife, Bénédicte, who agreed that I leave my job to become a student again (as long as I did not go to live in a "kot" with other students in Leuven). Many thanks to her. My thoughts also go to my twin brother and companion scientist, Hubert, who has a more "beautiful mind" than I have, and to my children, Simon (who led the way by earning his PhD in applied mathematics before me), Alice, and Quentin, for their encouragements.

Abstract

Many practical, real-world problems can be understood as the search for a state of affairs that extends a fixed state of affairs, which we call the "environment", with decisions that satisfy some given conditions and, optionally, maximize a deterministic benefit function. Such problems are found in, e.g., engineering, law or economics. The extended state of affairs, i.e., the "solution", describes a desirable course of action. "Configuration Problems" are examples of such problems.

The Knowledge Representation (KR) methodology to solve such problems is to:

(a) design a vocabulary Σ that allows representing the objects and their properties in the solution state of affairs,

(b) express the conditions to be satisfied as a logic sentence over vocabulary Σ, using an appropriate logic-based KR language, and

(c) submit the sentence and the representation of the environment to a generic algorithm for the *model expansion* inference to obtain a representation of the solution.

We contribute to this KR methodology in three ways.

First, we propose a standard for the KR language called FO($\cdot$) (aka FO-dot). FO($\cdot$) extends first-order logic with language constructs that make it more expressive for practical applications, namely: types and identifiers, data theories, arithmetic and aggregates, (possibly inductive) definitions, partial functions, and intensional objects (a new extension that we propose). We then developed a reference implementation of model expansion for FO($\cdot$), called IDP-Z3. IDP-Z3 improves on previous work by supporting arithmetic operations over the infinite set of numbers, opening up new fields of applications.

This standard KR approach is appropriate only when the conditions to be satisfied and the environment are fully known at the start of the search. Interactive model expansion has been developed to address the first limitation: in this approach, an interactive system assists the user in finding a solution that is acceptable to him. Our work addresses the second limitation: in many cases, the environment is not fully known at the start of the search, but can be observed during the search when needed. During the search, the user may consider tentative solutions that have environmental prerequisites. To ensure that the solution is appropriate, these prerequisites must be verified by observing the environment. It is thus important to distinguish fixed but unknown *environmental* variables from undecided *decision* variables. We also make a distinction between the laws of possible environments and the laws of acceptable solutions. We analyze the consequences of these distinctions on the knowledge representation and on the reasoning required to solve model expansion problems interactively, and we evaluate our implementation.

Finally, solutions of some model expansion problems have many symmetries, in the sense that some objects in those solutions can be exchanged to obtain another solution of the problem. We observe that symmetries can sometimes be used to compress structures onto a smaller domain without loss of information. This allows us to propose a novel method, that we call "lifted model expansion", that takes advantage of this compression to accelerate the search for solutions. We experimentally show its benefit for solving Social Golfer problems (which are commonly-used benchmarks for symmetry breaking techniques) and representative configuration problems from the manufacturing industry. These results encourage further research in the automatic reformulation of model expansion problems for speed gain.

Beknopte samenvatting

Veel praktische, echte problemen kunnen worden begrepen als het zoeken naar een stand van zaken die een bestaande toestand, de "omgeving" genaamd, uitbreidt met beslissingen die voldoen aan bepaalde voorwaarden en, optioneel, een profijtfunctie maximaliseren. Zulke problemen komen voor in bijvoorbeeld engineering, recht of economie. De uitgebreide toestand, d.w.z. de "oplossing", beschrijft de keuzes die tot dit optimum leiden. Zulke problemen komen voor in bijvoorbeeld engineering, recht of economie.

Binnen kennisrepresentatie (KR) is er voor dergelijke problemen een methodologie die bestaat uit drie stappen:

- Ontwerp een woordenboek Σ waarmee de objecten en hun eigenschappen kunnen worden voorgesteld in een toestand die een oplossing is.
- Beschrijf de omgeving en de voorwaarden waaraan voldaan moet worden als een logische zin in een op logica gebaseerde KR-taal met behulp van de terminologie uit het woordenboek Σ.
- Gebruik van een generiek algoritme voor modeluitbreiding-inferentie om een toestand te vinden die voldoet aan de logische zin; deze toestand stelt de oplossing voor.

We dragen op drie manieren bij aan deze KR methodologie.

Ten eerste presenteren we een standaard voor de FO($\cdot$) KR-taal (ook bekend als FO-dot). FO($\cdot$) breidt de eerste-orde logica uit met taalconstructies die het expressiever maken voor praktische toepassingen, namelijk: types en identifiers, datatheorieën, wiskundige bewerkingen en aggregaten, (mogelijk inductieve) definities, partiële functies, en intensionele objecten (een nieuwe uitbreiding die we voorstellen). Vervolgens hebben we een referentie-implementatie van modelexpansie ontwikkeld voor FO($\cdot$), genaamd IDP-Z3.

IDP-Z3 verbetert eerder werk door rekenkundige bewerkingen over de oneindige getallenverzameling te ondersteunen, wat een nieuw toepassingsgebied opent.

Vervolgens merken we op dat de informatie over de omgeving vaak niet volledig bekend is aan het begin van de zoekopdracht; in plaats daarvan kan de omgeving, waar nodig, tijdens de zoekopdracht worden geobserveerd. Tijdens het zoeken kan de gebruiker voorlopige oplossingen overwegen die uitgaan van een bepaalde toestand voor een aspect van de onbekende omgeving. Om er zeker van te zijn dat de oplossing geschikt is moet de gebruiker nakijken dat dit aspect overeen komt met de werkelijkheid. Het is dus belangrijk om onderscheid te maken tussen variabelen die een onbekend aspect van de omgeving voorstellen en variabelen die een keuze van de gebruiker voorstellen. We maken ook een onderscheid tussen de wetten waaraan de omgeving moet voldoen en de wetten waaraan een oplossing moet voldoen. We analyseren de gevolgen van deze onderscheiden op de kennisrepresentatie en op het redeneren dat nodig is om modeluitbreidingsproblemen interactief op te lossen, en we evalueren onze implementatie.

Tot slot hebben oplossingen van sommige modeluitbreidingsproblemen veel symmetrieën, in de zin dat sommige objecten in die oplossingen kunnen worden uitgewisseld om een andere oplossing van het probleem te verkrijgen. We stellen vast dat symmetrieën soms kunnen worden gebruikt om een structuur samen te drukken tot een kleinere structuur. Op basis hiervan hebben we een nieuwe methode ontwikkeld, die we “lifted model expansion” noemen, die gebruikmaakt van dit samendrukken om het zoeken naar oplossingen te versnellen. We tonen experimenteel het voordeel aan in representatieve configuratieproblemen uit de productie-industrie.

Contents

Abstract iii

Beknopte samenvatting v

Contents vii

Abbreviations xi

List of Symbols xiv

List of Figures xv

List of Tables xvii

List of Listings xix

1 Introduction 1
- 1.1 Motivating problems . . . 1
  - 1.1.1 Railway Network . . . 2
  - 1.1.2 Real Estate Sales Tax . . . 2
  - 1.1.3 Rack configuration . . . 3
- 1.2 First-order logic . . . 4
- 1.3 FO(ID) . . . 6

2 Standard Model Expansion 11
- 2.1 State of the art . . . 11
- 2.2 A proposed standard for FO(·) . . . 16
- 2.3 IDP-Z3, a reasoning engine for FO[.] . . . 24
- 2.4 Discussion . . . 31
- 2.5 Personal contribution . . . 33

3 Interactive Model Expansion 35

3.1 Introduction . . . 35
3.2 State of the art: the KB paradigm for product configuration . . 37
3.3 Theoretical contribution . . . 40
3.3.1 Split model expansion . . . 40
3.3.2 Interactive model expansion . . . 41
3.3.3 Propagation . . . 44
3.3.4 Partial solutions . . . 45
3.3.5 Relevance . . . 47
3.4 Implementation: the Interactive Consultant . . . 49
3.5 Experimental evaluation . . . 52
3.6 A generalization: Collaborative and Temporal Model Expansion 54
3.7 Discussion . . . 61
3.8 Personal contribution . . . 64

4 Lifted Model Expansion 65
4.1 Introduction . . . 65
4.2 State of the art: symmetry breaking . . . 68
4.3 Theoretical contribution . . . 70
4.3.1 Lossless compression of structures . . . 70
4.3.2 Translation into a lifted sentence . . . 77
4.3.3 Equisatisfiability . . . 82
4.4 Experimental evaluation . . . 95
4.4.1 Pigeonhole problems . . . 96
4.4.2 Generative configuration problems . . . 99
4.4.3 Social Golfer problem . . . 107
4.5 Discussion . . . 112
4.6 Personal contribution . . . 114

5 Conclusion 115
5.1 Future research directions . . . 115
5.2 Application: Intelligent spreadsheets . . . 115
5.3 Application: Intelligent chatbots . . . 116

A Partial functions 119
A.1 Personal contribution . . . 129

B Transformation to lift FO(Type, Aggregate) sentences 131

C Lifted theory for the Social Golfer problem 133

List of publications 135

Bibliography 137

Alphabetical Index **149**

Abbreviations

ASP	Answer set programming
BAPA	Boolean Algebra of sets with Pressburger Arithmetic
CSP	Constraint Satisfaction Problem
CP	Constraint Programming (solver)
CWA	Closed World Assumption
DCA	Domain Closure Axiom
FO	First-order logic
FO(ID)	First-order logic extended with inductive definitions
GCP	Generative configuration problems
IC	Interactive Consultant
IDP	a reasoning engine for FO($\cdot$)
KB	Knowledge Base
KR	Knowledge Representation
lcm	Least common multiplier
Lcm	Least common multiplier of the multiplicity of lifted elements
mul	Multiplicity
Mul	Multiplicity of lifted element(s)
MX	Model expansion problem
NF	Normal Form (of a formula)
PF	Partial function
SAT	Satisfiability (solver)
SMT	Satisfiability Modulo Theories (solver)
SMX	Split model expansion problem
SO	Second-order logic
UNA	Unique Name Axiom
webIDE	Online Integrated Development Environment
Z3	an SMT solver developed by Microsoft

List of Symbols

Vocabulary

Σ	a vocabulary, i.e., a set of type, predicate and function symbols
σ	a symbol
T	a type symbol
$\bar{T}$	a tuple of type symbols
$\mathbb{B}$	the set of Booleans
$\mathbb{Q}$	the set of rationals
p	a predicate symbol
f	a function symbol

Expression

ϵ	an expression
t	a term
$\bar{t}$	a tuple of terms
ϕ	a formula
$\neg$	not (negation)
$\wedge$	and (conjunction)
$\vee$	or (disjunction)
$\leftarrow$	definitional implication, or rule operator
$\forall$	universal quantification

$\exists$ existential quantification

Structure

I a structure, i.e., a domain and an interpretation of symbols

$dom(I)$ the domain of structure I

σ^I the interpretation of symbol σ in structure I

$[\![\epsilon]\!]^I$ the value of expression ϵ in structure I

$I \models \phi$ I is a model of ϕ, I satisfies ϕ

$I \leqslant_p J$ structure I is less precise than structure J; I can be expanded into J

$\widehat{I}$ the set of expansions of structure I, i.e., $\{J \mid I \leqslant_p J\}$

I_{env} a structure representing an environment

I_{obs} a structure representing the observations (less precise than I_{env})

I_{sol} a structure representing a solution

I_{dec} a structure representing (tentative) decisions (less precise than I_{sol})

Permutation

π a permutation

π^0 the identity permutation

$o_\pi(d)$ the π-orbit of element d; $\{\pi^i(d) \mid i \in \mathbb{N}\}$

Lifted theory

L a lifted structure

l an element in the domain of the lifted structure L

d an element in the domain of the concrete structure I

$\overline{\chi(\phi)}$ the translation of sentence ϕ

$\chi(\epsilon)$ the transformation of expression ϵ (a part of the translation)

List of Figures

1.1 Railway problem 2
1.2 Rack solution for 1 element of type B 3

3.1 Screenshot of the Interactive Consultant 49

4.1 Concrete solution of the Pigeonhole problem 66
4.2 Lifted solution. The number in a node is the multiplicity 66
4.3 Automorphisms with and without a backbone 72
4.4 Simple model illustrating the special rule for sum aggregates . 93
4.5 Wall clock time to solve our pigeonhole problem 96
4.6 Wall clock time to solve our "2-2 pigeonhole" problem 97
4.7 Wall clock time to solve a pigeonhole optimization problem . . 99
4.8 Lifted solutions for a pigeonhole optimization problem 99
4.9 Lifted configuration for the Rack ABCD 12 problem 104
4.10 Partial expansion of the Rack ABCD 12 solution 104
4.11 Time to solve Rack ABCD with symmetry breaking 106
4.12 Solution for the 1-4-1 Social Golfer problem 109
4.13 Lifted solution for the 8-4-1 Social Golfer problem 111
4.14 Wall clock time to solve some Social Player problems 111

5.1 VAT calculation in Excel 116

List of Tables

2.1 The abstraction relation 13
2.2 The modules of FO[·] 17
2.3 Resolution of binary quantifications and aggregates for $p : \bar{T} \to \mathbb{B}$ 26
2.4 Grounding of quantifications and aggregates 27
2.5 Translation of `{ P ← Q. Q ← P.}` 29
2.6 Translation of `{ P ← ¬Q. Q ← ¬P.}` 30
2.7 Comparison of language features of solvers. 32

3.1 Graphical cues in the Interactive Consultant 50
3.2 The number of observations and decisions is greatly reduced with our proposal. 53
3.3 Correspondence between Interactive Model Expansion and Temporal Model Expansion 59
3.4 Comparison of reasoning capabilities of solvers. 61
3.5 Various performance metrics of IDP-Z3. 63

4.1 Negation normal form of formulas and terms 78
4.2 Transformation of formulas and terms in negation normal form 80
4.3 Special rules for the transformation of sentences 81
4.4 Time to solve configuration problems 105

A.1 Knowledge in the domain signatures 128

B.1 Transformation of formulas and terms 132

List of Listings

1.1 Reachability relation 7

2.1 Vocabulary for crosswords 21
2.2 Knowledge base for the Railway problem 24
2.3 Solution of the Railway problem 25
2.4 Connected relation 27
2.5 Level mapping for a simple definition 28
2.6 Length of a list 30

3.1 Split knowledge base for Real Estate Sale Tax 40
3.2 Temporal vocabulary for a monthly nurse rostering problem . . 55
3.3 Temporal vocabulary for the Real Estate Sales Tax problem . . 60

4.1 Knowledge base for the Rack problem 100
4.2 4-4-2 Social Golfer problem 107

C.1 Lifted theory for the 8-4-2 Social Golfer problem 133

Chapter 1

Introduction

We begin by presenting the class of problems that motivate our research. We then recall the main concepts of first-order logic, and of one of its extension, called FO(ID), to support inductive definitions.

1.1 Motivating problems

One important aspect of the human mind is the ability to consider different possibilities, i.e., different "states of affairs", created by our imagination. According to Wittgenstein's Tractatus Logico-Philosophicus, "[States of affairs] have objects and properties as constituents. (...) the totality of states of affairs exhausts the space of possibilities" (Textor, 2021). Considering different states of affairs is often necessary to solve complex problems.

Indeed, many practical, real-world problems can be understood as the search for a state of affairs that extends a fixed state of affairs, which we call the "environment", with decisions that satisfy some given conditions and, optionally, maximize a deterministic benefit function. The extended state of affairs, i.e., the "solution", describes a course of action that makes the world better (at least for some people). Such problems are found in, e.g., engineering, law or management.

Example 1. *A toy example of such problems is the pigeonhole problem: the environment is a set of pigeons and pigeonholes; the solution is the same set of pigeons and pigeonholes, together with an assignment of each pigeon to one pigeonhole, so that each pigeonhole is not assigned more than one pigeon.*

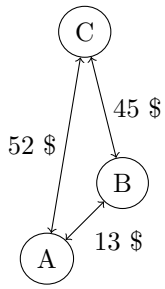
Problems of that form are the motivation and the focus of this thesis. A key feature of such problems is that the given state of affairs is fixed and that the satisfaction of the given conditions only depends on that given state of affairs and on decisions that are freely made by an agent (or by a set of collaborating agents, as discussed in Section 3.6). More complex problems involving competing agents are not in our scope.

Below are three motivating problems that will be used to illustrate concepts and methods in each of the three following chapters.

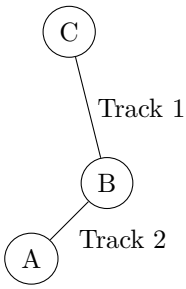
1.1.1 Railway Network

The environment consists of a list of cities, and of a survey detailing the cost of building each possible rail track linking two cities (each rail track links exactly one city to exactly one other city, for both directions). The solution is the environment extended with a rail track network such that every city is reachable from every other city. The total cost is the sum of the cost of the rail tracks in the railway network. The benefit is maximized when the total cost is minimized.

An environment with 3 cities, and its solution, are illustrated in Figure 1.1.



(a) Environment



(b) Network in the solution

Figure 1.1: Railway problem

1.1.2 Real Estate Sales Tax

This example is a simplified version of the real estate sales tax problem discussed by Deryck et al. (2019).

The environment consists of a real estate property that has been sold. The solution is the environment extended with a registration to the tax authority.

The registration must abide by the Real Estate Sales Tax law. The law states the default tax rate, as well as various tax rebates that can be claimed when their preconditions are satisfied. The registration states the tax rebate claimed, if any, and the documentation to prove that its preconditions are met. The cost is the tax amount to be paid to the tax authorities. The benefit is maximized when the cost is minimized.

The tax amount is 10 % of the sales price for standard registration with the authorities, but a lower tax rate can be obtained when the house is bought to be leased to tenants for a low rent, or when the seller is licensed to sell social housing.

1.1.3 Rack configuration

The Rack Configuration problem has been presented as a typical hardware configuration problem found in industry (Comploi-Taupe et al., 2022).

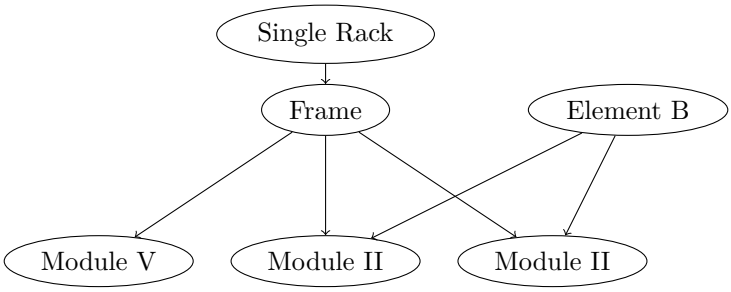


Figure 1.2: Rack solution for 1 element of type B

The environment consists of a set of elements. Each element is of type A, B, C or D. The solution is the environment extended with a set of interrelated racks, frames and modules. There are two types of Racks: Single or Double; there are 5 types of modules: I, II, III, IV and V. The solution must satisfy these requirements:

- each module is contained in exactly one frame; each frame is contained in exactly one rack;
- a rack Single can contain up to 4 frames; a Double up to 8; a frame can contain up to 5 modules;
- a module controls 0 or 1 element; an element A (resp. B, C, D) is controlled by 1 (resp. 2, 3, 4) module of type I (resp. II, III, IV);

- a frame can contain a module II if and only if it contains exactly one module V.

A solution for 1 element of type B is illustrated in Figure 1.2 on the preceding page.

Notice that the number of racks, frames and modules in the solution is not known: a part of the problem is to find that number.

1.2 First-order logic

We now recall the main concepts of classical first-order logic (FO).

Vocabulary and formula A *vocabulary* Σ is a set of predicate and function symbols. Each predicate and function symbol has an arity (a positive number). Symbols with arity zero (resp. one) are *nullary* (resp. *unary*) symbols.

A Σ-*term*, denoted by t, is an expression consisting of a variable, or of a function symbol of vocabulary Σ applied to Σ-terms. We simply say "a term" when the vocabulary is clear from the context. The terms applied to a function are called the arguments of the function application. The number of terms applied to a function symbol must be equal to its arity. A *constant* is a function of arity 0 applied to an empty tuple, e.g., $c()$, or, simply, c.

A Σ-*atom* is an expression consisting of a predicate symbol in vocabulary Σ applied to Σ-terms. The number of Σ-terms applied to a predicate symbol must be equal to its arity. A Σ-*formula*, denoted ϕ, is an expression built using Σ-atoms, logic connectives, and universally $(\forall x : \psi)$ and existentially $(\exists x : \psi)$ quantified formula. We say that x is declared by the quantification, and that occurrences of x in ψ are in the scope of the quantifier. The logic connectives are $\neg$ (negation), $\wedge$ (conjunction, "and"), $\vee$ (disjunction, "or"), $\Rightarrow$ (material implication), and $\Leftrightarrow$ (equivalence). Again, we simply say "an atom" (resp. "a formula") when the vocabulary is clear from the context.

For simplicity, we assume that a variable is declared only once in a formula, by a quantification. The *free* variables of an expression ϵ, denoted $vars(\epsilon)$, are the variables occurring in ϵ that are not in the scope of a quantifier that declares them in ϵ. A ground expression is an expression without free variables. A *fact* is a ground atom. A *sentence* is a ground formula. A *theory* is a set of sentences.

Structure A *Σ-structure* (also known as *interpretation*) I, consists of a *domain of discourse* (or simply, a domain) and an *interpretation* of each symbol of vocabulary Σ:

- the domain of discourse (denoted D) is a nonempty set of elements called *domain elements*;
- the interpretation p^I of a predicate symbol (resp. f^I of a function symbol) of arity n is a function from D^n to the booleans $\mathbb{B}$ (resp. to D).

The interpretation of a predicate symbol is represented by the set of tuples $\bar{d}$ of domain elements that make p^I true. The interpretation of a function symbol is represented by a set of pairs $(\bar{d}, d)$, also denoted by $\bar{d} \mapsto d$, where $d = f^I(\bar{d})$.

Again, we simply say "a structure" when the vocabulary is clear from the context.

A *total structure* is a structure in which the functions interpreting predicate and function symbols are total; a *partial* structure is a structure that is not total. A partial Σ-structure I_2 is more precise than partial Σ-structure I_1, denoted by $I_2 \geqslant_p I_1$, if and only if they have the same interpretation of the types, and the functions interpreting the predicate and function symbols in I_2 are more precise than those interpreting them in I_1 (a function f_2 is more precise than f_1 if and only if (i) the domain of definition of f_2 includes the domain of definition of f_1, and (ii) an element of the domain of definition of f_1 has the same image by f_1 and f_2). In this case, we say that I_2 is an *expansion* of I_1.

Formula interpretation We denote by $I[\bar{x} : \bar{d}]$ a structure I extended with the *variable assignment* $[\bar{x} : \bar{d}]$, i.e., with the mapping from the tuple of variables $\bar{x}$ to a tuple of domain elements $\bar{d}$. The value of an expression (term or formula) in a total structure extended with a variable assignment for its free variables is defined as follows:

- the value of a variable is the domain element assigned to it by the variable assignment;
- the value of a term $f(\bar{t})$ is the value defined by f^I for the values t_i^I of the arguments (this requires the interpretation of function symbols to be total functions);
- the value of an atom $p(\bar{t})$ is the value defined by its interpretation p^I for the values t_i^I of its arguments;

- the value of a formula is derived from those of the involved atoms by the standard evaluation of the logic connectives;
- the value of an existentially (resp. universally) quantified formula is true if and only if the inner formula is true for at least one (resp. for all) possible assignment of the quantified variable.

We denote the value of a formula ϕ in a total, extended structure $I[\bar{x} : \bar{d}]$ by $[\![\phi]\!]^{I[\bar{x}:\bar{d}]}$ (or, for brevity, by $\phi^{I[\bar{x}:\bar{d}]}$). Note that the value of a sentence is independent of any variable assignment.

Two expressions are (logically) equivalent if they have the same value in any structure extended with any variable assignment for the union of the set of their free variables.

A total structure I is a *model* of a sentence ϕ if the value of the sentence is **true** in the structure, i.e., if it *satisfies* it. We write: $I \models \phi$. The problem of satisfiability checking is the problem of deciding whether a sentence has a model.

1.3 FO(ID)

Definitions are an important, common, and precise form of human knowledge. Definitions define one or more *defined concepts* in terms of other *parameter concepts*, meaning that they specify a unique value for the defined concept(s), given a value for the parameter concepts. For example, the previous section provided the definitions of terms and formulas, given a vocabulary.

It is well known that first-order logic cannot represent so-called *inductive definitions*[1], such as the reachability relation of a graph (i.e., the set of pairs of (indirectly) connected nodes). By contrast, the extension of first-order logic proposed by Denecker (2000), called FO(ID), can represent such definitions (see Example 2).

Syntactically, a *definition* is a set of *rules* of the form $\forall \bar{x} : \mathit{head} \leftarrow \mathit{body}$, where *head* is an atom and *body* a formula, both with free variables $\bar{x}$, and where $\leftarrow$ is the *definitional implication* (or *rule operator*). The parameter symbols of a definition are the non-defined symbols that occur in the bodies. An *inductive definition* is a definition in which a defined symbol occurs in a body of a rule.

[1] unless additional "level mapping" symbols are introduced, as we'll see in Section 2.3 on page 24

Example 2. *The reachability relation for the Railway problem of Section 1.1.1 on page 2 is defined in typed FO(ID) in Listing 1.1:*

Listing 1.1: Reachability relation

```
{ ∀c ∈ City: reachable(c,c).
  ∀c1,c2 ∈ City: reachable(c1,c2) ←
       railtrack(c1,c2) ∨ railtrack(c2,c1).
  ∀c1,c2 ∈ City: reachable(c1,c2) ←
       ∃c3 ∈ City: reachable(c1,c3) ∧ reachable(c3,c2).
}
```

`reachable` *is the defined symbol;* `railtrack` *is not defined but occurs in the body of the second rule: it is the parameter of the definition. Because* `reachable` *occurs in the body of the third rule, the definition is inductive.*

Logic programs also take the form of a set of rules. Several semantics have been proposed to define their formal meaning. For example, Clark's completion semantics (Clark, 1977) proposes that a head symbol has the same truth value as the disjunction of the bodies of the rules having that head. Two frameworks view logic programs as inductive definitions: the Justification Theory (Denecker et al., 2015), and the Approximation Fixpoint Theory (Denecker et al., 2012). The Justification Theory uses justification graphs to represent why certain facts hold (or do not hold) in a model; Approximate Fixpoint Theory uses approximating operators on a bilattice of structures to represent the inductive process. We refer the reader to Marynissen et al. (2021) for a comparison of these two frameworks.

Some semantics of logic programs, e.g., the stable semantics, do not view sets of rules as definitions: as a consequence, a logic program may have several stable models. By contrast, FO(ID) takes the view that such programs are definitions.

Informally, a set of rules is interpreted as an (inductive) definition. This means that a rule is not interpreted merely as a material implication, but as a *production*. The set or relation (possibly more than one) defined by the set of rules is obtained as the limit of the *induction process*. This induction process constructs the defined objects from the interpretation of the parameters of the definition, by iterative rule application, as follows. At each stage, one has an interpretation for the parameter and defined symbols. One starts with an empty interpretation of the defined symbols, and the given interpretation for the parameter symbols. Each rule is instantiated for each possible assignment of its head variables, resulting in rule *instances*. This instantiation is finite when the domain of the symbols is finite. A rule instance is *applicable* at a stage i of the iterative process when its body is true given the interpretations of the

symbols. At stage i, one *applies* an applicable rule instance, i.e., one adds the head of an applicable rule instance to the interpretation of the defined symbols, to obtain the interpretation of the defined symbols for the next stage. This process is repeated until a fixpoint is reached. A fixpoint is always reached when the domains of the defined symbols are finite. Then, the interpretation of a defined symbol is its fixpoint.

Example 3. *For the definition of reachability above, initially, the interpretation of* `reachable` *is the empty set; then, every pair of identical cities (rule 1) and every pair of cities directly linked by a rail track (rule 2) are added to its interpretation. Adding pairs of cities to this interpretation may make new pairs of cities reachable by the third rule. The third rule is applied repetitively until it does not add any new pair of cities: the set of pairs of cities obtained at that point is the interpretation of* `reachable`*.*

When the definition is not inductive, the order in which the rule instances are applied in the iterative process, i.e., the *order of rule application*, does not matter: any order will reach the same fixpoint and yield the intended interpretation.

When the definition is inductive, but its rule instances are applicable only by the presence of elements in the defined relation (as in the reachability example above), the definition is *monotone*: in that case, the order of rule application does not matter either. When a defined atom occurs negated in the body of a rule (in Disjunctive normal form), the *absence* of elements may make a rule instance applicable: the definition is *non-monotone*, and the order of application does matter.

When the definition is non-monotone, then the order of rule application matters. A definition is *stratified* if each defined atom can be mapped to a natural number, called the *level of the atom*, such that (i) the level of all atoms in the body of any rule is smaller or equal than the level of the head atom, and (ii) if the atom occurs negatively in a body, its level is strictly smaller than the level of the head atom.. We say that defined atoms having the same level are in the same stratum. In that case, a safe order of rule application is obtained by following the stratification, i.e., by applying rules deriving atoms with smaller level before applying rules deriving atoms with a higher level. This ensures that a rule is applied only when it is "safe" to do so, i.e., when the body of the rule will continue to be true as the iterative process continues (Bogaerts et al., 2018). Any process of rule application that applies rules in such a safe way reaches the same fixpoint, irrespective of the order of rule application within each stratum, and this fixpoint is indeed the intended interpretation.

Some non-monotone definitions cannot be stratified, however. In that case, the

order of rule application must still respect an *induction order* to ensure that the value of defined atoms is determined using a rule instance only when it is safe to do so. An induction order is a well-founded order on the defined atoms, laying constraints on the application of rules: if two rule instances are applicable and one derives an atom strictly smaller in the induction order than the atom derived by the other rule, then the first rule instance must be applied earlier (Denecker & Vennekens, 2014). This well-founded semantics gives inductive definitions a formal meaning that matches our intuition.

Example 4. *The truth value of a formula in a structure is defined by induction on the structure of formulas. The induction order of that definition is such that a subformula of formula ϕ is smaller than formula ϕ. Thus, one must always apply a rule instance inferring the value of ψ before applying another rule instance inferring the value of formula ϕ if ψ is lower than ϕ in the induction order, i.e., if ψ is a subformula of ϕ.*

The definition of functions also presents some difficulties. Like inductive definitions of predicate, recursive definitions of functions are definitions in which the defined symbol occurs in the body of a rule of the definition. Typical examples include the definition of the length of a list, or of the factorial of a natural number.

Because a function of arity n can be seen as a predicate of arity $n + 1$, called the *graph* of the function, with an unicity constraint on its last argument, a recursive definition can sometimes be re-written as an inductive definition of its graph predicate. But this is challenging when the defined symbols occur in compound terms in the body of a rule, e.g., in an aggregate expression, as in the following example.

Example 5 (Company control)**.** *Consider the informal statement "every American company is controlled by American companies", where "control" is defined as follows: "A company A controls a company B if the total sum of the shares in company B owned by A, or by companies controlled by A, is more than 50%."*

We use the following vocabulary:

- *predicate Cont: $Cont(x, y)$ is informally interpreted as "company x controls company y";*
- *function OwnsSh: $OwnsSh(x, y) = s$ is informally interpreted as "company x owns s % of the shares of company y";*
- *predicate American: $American(x)$ is informally interpreted as "company x is American".*

The first FO(ID) formula below defines Cont and the second expresses the statement above:

$$\{\forall a \forall b(Cont(a,b) \leftarrow 50 < \sum_{c|c=a \vee Cont(a,c)} OwnsSh(c,b))\} \tag{1.1}$$

$$\forall c, c1(Cont(c,c1) \wedge American(c1) \Rightarrow American(c)) \tag{1.2}$$

Notice that the defined symbol Cont occurs in the filter of the sum aggregate: the definition is inductive.

Nested justification systems can be used to define the formal semantics of such inductive definitions (Marynissen et al., 2022).

The head of a rule in FO(ID) must be a single atom, in contrast to ASP which allows a disjunction in the head of so-called choice rules. However, an extension of FO(ID), called FO(C), introduces disjunctions in the head, as well as other language constructs, to describe causal processes (Bogaerts et al., 2014). Unlike in default logic (Reiter, 1980), exceptions to an FO(ID) rule must be explicitly stated in the rule (possibly in the form of a predicate defined separately). Unlike in defeasible logic (Nute, 2001), FO(ID) rules do not have any priority ordering.

Chapter 2

Standard Model Expansion

We present the "standard model" expansion method to solve the motivating examples, and our contribution, i.e., a proposed standard for FO(·), and a reference implementation for this standard, called IDP-Z3.

2.1 State of the art

While the motivating problems of Section 1.1 on page 1 can be solved by dedicated algorithms implemented using traditional programming languages, they can also be solved by generic algorithms acting on symbolic representations of the problem. The latter approach is based on "Knowledge Representation", and is usually cost-effective because it usually reduces the amount of labor required to solve the problem and facilitates the re-use of sophisticated search algorithms.

The Knowledge Representation (KR) methodology consists of the following steps:

(a) design a vocabulary Σ that allows representing knowledge about states of affairs,

(b) represent the general knowledge about states of affairs in this vocabulary Σ, i.e., express the laws that are true in the states of affairs under consideration, using an appropriate logic-based KR language,

(c) use the proper inference to solve the problem at hand.

The inference used to solve the problems motivating this thesis is the *model expansion* inference (Mitchell et al., 2006): its input is the representation of the knowledge of the problem domain and the representation of the environment; its output is the representation of a solution. "Satisfiability checking", "model search", "constraint solving" and "optimization" denotes similar methods and algorithms.

By this methodology, in step (a), a suitable vocabulary is chosen. It consists of a set of symbols representing concepts. A symbol is either a type or a relation symbol. A type symbol is characterized by its informal meaning: a type symbol represents the concept of a category of objects in a state of affairs. A relation symbol is characterized by its informal meaning, by a number of arguments, and by a type for each of its arguments. A relation symbol represents the concept of a relation between different objects of appropriate types in a state of affairs.

Usually, we choose evocative names for symbols so it is clear what they abstract. Note that an argument in a relation may have a functional dependency with other arguments of the relation. In that case, it is beneficial to represent the relation as a function.

Example 6. *A possible vocabulary for the Railway network problem consists of the type symbols* `City, Cost`*, and of these relation symbols:*

```
railtrack(City, City)
cost(City, City, Cost)
totalCost(Cost)
```

The meaning of `railtrack(x,y)`*, where* `x,y` *are cities, is that the railway network has a rail track between* `x` *and* `y`*. The meaning of* `cost(x,y,c)`*, where* `x,y` *are cities and* `c` *is a cost, is that the cost of building a rail track between* `x` *and* `y` *is* `c`*. The meaning of* `totalCost(c)`*, where* `c` *is a cost, is that the cost of building the railway network is* `c`*.*

The following vocabulary uses functions to represent the functional dependencies between arguments:

```
railtrack(City, City)
cost : (City, City) → Cost
totalCost : () → Cost
```

For step (b), i.e., to express the general knowledge about states of affairs, first-order logic is attractive as a language because it has a clear and mathematically precise formal semantics (see Enderton, 1972), and a simple and natural informal semantics:

- The formal semantics of the logic connectives $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow, \forall, \exists$ are clear, precise, simple and natural. The formal semantics of FO defines the set of structures in which a *(logic) sentence* is true, i.e., it defines the *satisfaction* relation.
- The informal semantics of FO extends the *abstraction* relation (Table 2.1): a sentence in FO is the abstraction of a sentence in a natural language (e.g., English); a structure satisfying a sentence, i.e., a *model*, is the abstraction of a state of affairs in which the sentence in natural language is true.

... is the abstraction of ...	
A symbol	a concept
A structure	a state of affairs
An element in the domain	an object in the state of affairs
The interpretation of a symbol	a property or relation of object(s)
A (logic) sentence	a sentence in natural language

Table 2.1: The abstraction relation

First-order logic can be used successfully for the modelling of some aspects of the physical world. Its formal and informal semantics is essential to its success. The abstraction relation maps logical results to a physical reality. A theory, i.e., a set of FO sentences, represents knowledge about the physical world by partitioning the set of states of affairs in two sets: the physically possible ones (their abstraction satisfies all the sentences in the theory), and the physically impossible ones (their abstraction does not). The more knowledge a theory contains, the fewer models it has, and the fewer states of affairs are deemed physically possible. Modelling of the physical reality is performed by adding sentences to the theory, i.e., by reducing the number of its models. Modelling is complete when the set of models is precisely the set of abstractions of the physically possible states of affairs.

Many reasoning methods and tools exist for FO. For example, deductive methods have been developed to find sentences entailed by a theory, or to prove that two theories are equivalent. These concepts of entailment and equivalence are defined in terms of the distinction between models and non-models. But the knowledge in a FO theory can be used to solve many other computational problems beyond computing entailment and equivalence. In this chapter, we use it for model expansion; in the next chapter, we shall explore many other uses.

Recall that, when I_2 is more precise than I_1, we say that I_2 is an *expansion* of I_1. The set of total expansions of I_1 is denoted by $\widetilde{I_1}$. We define model expansion and model optimization (step (c)) as follows:

Definition 1. *The* (standard) model expansion problem $\mathcal{MX}$ *is a problem that takes as input a tuple* (Σ, I_{env}, ϕ) *where* I_{env} *is a total structure over* $\Sigma_{I_{env}} \subseteq \Sigma$ *and* ϕ *a sentence over* Σ*. A* solution *for such a problem is a total structure* I *over* Σ *that expands* I_{env} *and satisfies* ϕ *:*

$$(I \in \widetilde{I_{env}}) \wedge (I \models \phi)$$

Definition 2. *The* (standard) model optimization problem $\mathcal{MO}$ *is a problem that takes as input a tuple* $(\Sigma, I_{env}, \phi, t)$ *where* I_{env} *is a total structure over* $\Sigma_{I_{env}} \subseteq \Sigma$*,* ϕ *a sentence over* Σ*, and* t *a term over* Σ*. A* solution *to such a problem is a solution* I *to* $\mathcal{MX}(\Sigma, I_{env}, \phi)$ *that minimizes* t*, i.e., :*

$$(I \in \widetilde{I_{env}}) \wedge (I \models \phi)$$

and

$$\forall I' \in \widetilde{I_{env}} : (I' \models \phi) \Rightarrow [\![t]\!]^I \leqslant [\![t]\!]^{I'}$$

Recall that we consider only problems without uncertainty: the satisfaction of the given conditions of the problem only depends on the fixed environment and on the decisions that are freely made. If this satisfaction also depended on additional variables not under the control of the decision maker, a model expansion problem could be solved for each possible interpretation of these variables to obtain a set of solutions, one for each eventuality. Such problems are studied in, e.g., game theory, which also discusses probabilistic strategies. Alternatively, the problem may be to find a solution that satisfies the conditions in all eventualities: such problems are called $\exists\forall SO$ satisfiability problems (Vlaeminck et al., 2012).

The sentence ϕ in the $\mathcal{MX}$ problem represents the knowledge of the *problem domain.* A triple $(\Sigma, \Sigma_{I_{env}}, \phi)$ defines a *model expansion problem class*, i.e., a set of model expansion problems having the same input vocabulary and sentence. The model expansion problem $\mathcal{MX}(\Sigma, I_{env}, \phi)$ is an *instance* of the model expansion problem class $(\Sigma, \Sigma_{I_{env}}, \phi)$.

If there are N symbols in the vocabulary Σ, there are 2^N model expansion problem classes for sentence ϕ. Two problem classes stand out:

- the one where $\Sigma_{I_{env}}$ is the empty set of symbols. We call $\mathcal{MX}(\Sigma, \{\}, \phi)$ a "model search" problem;

- the one where $\Sigma_{I_{env}} = \Sigma$. We then call $\mathcal{MX}(\Sigma, I_{env}, \phi)$ "model checking" problem.

An important aspect of knowledge representation is that the same knowledge of the problem domain, i.e., the same sentence ϕ, can be used to solve many different problems, and in particular, problems in different model expansion problem classes. In other words, the knowledge can be *reused*. This is in stark contrast with *programs* which essentially compute an output given the interpretation of the symbols in a pre-defined set of input symbols, i.e., solve instances of one particular problem class. The program has to be rewritten if the set of input symbols changes.

An often desirable property of a model expansion problem class is that there exists a solution I for every input I_{env}. For example, we expect the Railway Network problem (Section 1.1.1 on page 2) to be solvable for any list of cities, and for any estimated cost to link them. We say that such a problem class is "universally solvable". Informally, the conditions to be satisfied allow a course of action for any situation that could be faced.

Despite its success in formulating science, FO has many weaknesses for representing human expert knowledge in complex, real-world applications. For this reason, the FO(·) language (Denecker & Vennekens, 2008) has been proposed to extend FO with a range of language constructs such as types and identifiers, arithmetic and aggregates (Denecker et al., 2001), (inductive) definitions (Denecker, 2000), non-deterministic inductive construction (Bogaerts et al., 2014), partial functions (Markovic et al., 2023), and intensional objects (Carbonnelle et al., 2023b), to make it more expressive while keeping the essential properties of classical FO listed above. FO(·) defines the formal and informal semantics of these constructs by extending the definition of the classical satisfaction and abstraction relations.

FO(·) attempts to combine the strengths of FO and of logic programming. FO can represent assertions about states of affairs, but cannot represent inductive definitions. By contrast, logic programs can represent inductive definitions, but cannot represent assertions about states of affairs in a natural way (e.g. $\mathtt{p} \vee \mathtt{q}$).

Answer Set Programming (ASP, Lifschitz, 2008) is another well-known KR language. FO(·) and ASP differ in the semantics given to the rule operator and to the negation connective: FO(·) keeps the classical meaning of negation but uses a non-classical meaning of the rule operator; by contrast, ASP interprets the rule operator as classical material implication, but gives a non-classical meaning to negation, called "negation as failure" (Gelfond & Lifschitz, 1988).

We believe that FO(·) is superior to ASP for knowledge representation because

its syntax, formal and informal semantics are simple and natural. This is important not just for the convenience of the knowledge engineer. Just as artificial neural networks try to mimic biological ones, we see value in emulating the way we think. After all, our mental capabilities are the results of a long evolution that favored mental capabilities useful for survival in the real world, at low computational costs.

2.2 A proposed standard for FO(·)

The community of researchers working on SAT (Satisfiability), SMT (Satisfiability Modulo Theories) and ASP (Answer Set Programming) solvers have benefited from the standardization of their modelling language. The standards are called CNF-DIMACS, SMT-Lib2 (Barrett et al., 2010a) and ASP-core (Calimeri et al., 2012) respectively. The availability of a common language has accelerated the development of solvers, enabled the development of benchmarks to allow the comparison of their performance, and facilitated their adoption by industry.

As the FO(·) language is now supported by several reasoning engines (namely, IDP3 (De Cat et al., 2013), FOLASP (Deryck et al., 2019), and IDP-Z3 (Carbonnelle et al., 2022)), we thought that it is time to write a draft standard for FO(·), with the hope that it will be widely adopted. We call it FO[·] (with square brackets). Thus, FO(·) denotes the language as discussed in scientific papers, while FO[·] denotes the concrete syntax used to create knowledge bases for reasoning engines.

The standard specifies the syntax of sentences, including when they are well-formed, and their formal semantics. It is available online.[1]

The driving principle for its design is to adhere closely to the traditional mathematical notation that we are all familiar with. For example, `∀ x ∈ T: p(x)` represents "for all x in type T, p holds" in FO[·] (while the IDP3 version of FO(·) would use the unfamiliar notation `!x[T]: p(x)`).

Like the SMT-Lib2 language, the FO[·] language is modular: while FO[·] denotes the language with all the extensions, FO[x, y] denotes classical logic with extension x and y only (for example, FO[Int, Aggregate]). The extensions are shown in Table 2.2.

Future extensions may include:

[1] `https://fo-dot.readthedocs.io/`

Module	Description
Core	Typed logic with finite data structures and equality
Sugar	Variable declaration, binary quantification and other conveniences
ID	(Inductive) definitions
Int	Integer arithmetic, ranges and dates
Real	Real arithmetic
Agg	Aggregates: cardinality, sum, min, max
Infinite	Quantification over infinite domains, including recursive data types
PF	Partial functions
Concept	Quantification over intensional objects

Table 2.2: The modules of FO[·]

- FO[Causal]: to represent causal processes according to FO(C) (Bogaerts et al., 2014);
- FO[Unit]: to represent units of measures for physical quantities;
- FO[Cos]: to represent transcendental functions (e.g., *cos*).

We highlight the main features of FO[·] below.

Types and quantification In real applications, the universe tends to be heterogeneous, and there are virtually no properties or concepts shared by all objects in the domain. In the human mind, the universe is partitioned into different *categories* of objects, each with its own class of attributes and relations. A type system serves to support this view, leading to more natural and precise knowledge representation. (Types are also called "sorts" in the literature (Wang, 1952))

Besides boolean, integer and real built-in types, FO[·] allows the creation of custom types, e.g., `City`. Predicates and functions are declared in the vocabulary, with a *type signature*, e.g., `railtrack: City` × `City` → $\mathbb{B}$. In the quantified formula ∀`x` ∈ `T: p(x)`, `x` ranges over the extension of type `T`. Types are used to syntactically verify that formulas are *well-typed*, helping detect common errors. It is also convenient to quantify over the set interpreting a predicate, e.g., over the `railtrack` predicate: ∃`(c1,c2)` ∈ `railtrack: c1=A`. We call this *binary* quantification.

Variables can be declared in the vocabulary (`var x` ∈ `T`): this aligns with the traditional practice of choosing evocative variable names in quantifications, and simplifies the writing of quantifications without the need for type inference (∀ `x: p(x)`). Also in line with traditional practices, variables with a numeric

suffix (`x1, x2`) have the same type as the declared variable without the suffix (`x`).

Data theory In modelling a state of affairs, it is often useful to state that some terms are used as *identifiers* of domain elements (Unique Name Axiom, UNA), that the given interpretation of a type is complete (Domain Closure Axiom, DCA), or that the interpretation of a predicate symbol is complete (Closed World Assumption, CWA). Stating these axioms in FO is cumbersome. For example, the UNA consists of a number of ground atoms quadratic in the number of identifiers: it states that the values of any two distinct identifiers are distinct; asymptotically, the number of pairs of distinct identifiers is the square of the number of identifiers.

To address this issue, FO[·] theories may use enumerations. An enumeration defines the set of identifiers interpreting a type (under UNA and DCA), the set interpreting a predicate symbol (under CWA), or the interpretation of a function symbol. A structure abstracting a state of affairs is then a *data theory*, i.e., a theory containing only enumerations.

Example 7. *The following vocabulary `V` and data theory `S` describe the environment of the Railway problem.*

```
vocabulary V {
  type City ≜ {A, B, C}
  cost : City × City → Int
}
structure S:V {
  cost ≜ { (A,A)→ 0, (B,B)→ 0, (C,C)→0,
           (A,B)→13, (B,A)→13,
           (B,C)→45, (C,B)→45,
           (A,C)→52, (C,A)→52 }.
}
```

Identifiers differ from constants in that they are subject to UNA. To make that distinction explicit, FO[·] requires parenthesis after constants (e.g., `startingCity()`), but not after identifiers (e.g., `A`).

Identifiers may be compound terms: for example, a cell in a grid is identified by a row and a column. Compound identifiers can be specified using a *constructor*: `type Cell ≜ constructed from {cell(x: Int, y: Int)}`. Here, `cell` is a constructor, and `x, y` are *accessors*: term `x(c)` is the `x` position of cell `c`. The construction process can be recursive, e.g., to define the concept of a list of cities: `type List ≜ constructed from{ nil, cons(car:City, cdr: List)}`.

This declaration implicitly declares the predicates `is_nil, is_cons: Cell→𝔹`: atom `is_nil(x)` is true when `x` is a List constructed with `nil`.

Arithmetic FO[·] supports formulas with the 4 arithmetic operations (addition, subtraction, multiplication and division) on the infinite set of integers and reals (or rather, rationals), as well as the comparison operators (equality, disequality and inequalities). It does not support transcendental functions (yet). (`log,exp,sin,cos`, ...).

Cardinality and Aggregates The number of elements of finite type T that satisfy a property `p` is formulated in FO[·] as `#{x ∈ T: p(x)}`. Similarly, `sum{{f(x)| x∈ T:p(x)}}` denotes the sum of `f(x)` for each `x` in `T` that satisfy `p`. Similarly, the minimum and maximum aggregates are denoted `min`, `max`.

Definitions (inductive or non-inductive) Besides logic sentences and enumerations, an FO[·] knowledge base may include (possibly inductive) definitions of sets and relations, as described in Section 1.3 on page 6 in the Introduction chapter. However, nested justifications are not supported; hence, a defined symbol cannot occur in a nested term (or in an aggregate) in the body of a rule of its definition.

By default, the formal semantics of definitions is the well-founded semantics (Denecker & Vennekens, 2014), but other semantics can be used (namely the completion, stable, co-inductive and Kripke-Kleene semantics). The completion semantics has been described in Section 1.3 on page 6. The construction process for the co-inductive semantics assumes that the propositions are initially true, and made false by application of rules (Gupta et al., 2007). The stable semantics is the semantics used in Answer Set Programming (Gelfond & Lifschitz, 1988), under which a set of rules may have several models. The Kripke-Kleene semantics excludes models with an infinite loop in the justification tree (Fitting, 1985).

Partial functions In typed logic, the interpretation of the function symbol $f : \bar{T} \to T$ is a total function over the cross-product of the interpretation of the $\bar{T}$ types. In many practical KR applications, however, some functions are actually partial over the cross-product of their domain types. When a function is applied to arguments out of its domain, a non-denoting term results: it has no informal meaning, i.e., it has no correspondence with an object in the state of affairs.

Example 8. *Consider the term `spouseOf(Tom)`, where `Tom` is of type `Person`, as expected by the type signature of `spouseOf`. The term is non-denoting if Tom is*

not married, even though it is well-typed. In arithmetic expression, the term $\frac{x}{y-1}$ *is non-denoting when* $y = 1$*, even when* x *and* y *are of a numeric type.*

Sentences with non-denoting terms are ambiguous. Consider the sentence "the spouse of Tom is alive". According to Bertrand Russell, it could mean either that "there is a spouse of Tom who is alive", or "any person who is the spouse of Tom is alive": the first sentence is false when Tom is not married, while the second is true in that case. We say that such sentences are not *well-defined*.

According to good mathematical practice, non-denoting terms can only appear in contexts where they are irrelevant. For example, $\exists x, y : y \neq 1 \wedge \frac{x}{y-1} > 1$ is accepted, but not $\exists x, y : \frac{x}{y-1} > 1$. In the first sentence, $y \neq 1$ is used as a *guard*, and $\wedge$ is interpreted non-commutatively: if the value of $y \neq 1$ is false, the value of the conjunction is false, and the second conjunct is undefined, but irrelevant; if the order of the conjuncts were inverted, the value of the conjunction would be undefined.

FO($\cdot$) follows these good mathematical practices, and requires that non-denoting terms appear only in contexts where they are irrelevant, in any structure (Markovic et al., 2023). The verification method is described in Appendix A on page 119, and is inspired by Berezin et al. (2004). It has 2 steps: (i) each sentence ϕ in the theory is syntactically transformed to another sentence that is true in a structure I only when all terms in ϕ are denoting in I; (ii) the transformed sentence is checked to be a tautology, i.e., that it is true in all structures. By this method, the property of an expression being well-defined is a decidable property in FO[$\cdot$] (excluding non-linear arithmetic), and can cope with functions having an unknown domain (Markovic et al., 2023).

This requirement helps avoid subtle modelling problems, but can make the sentences more verbose. To avoid verbosity, FO[$\cdot$] introduces new language constructs in the declaration of symbols:

- `type` $\texttt{T}_i \sqsubseteq \texttt{T}_{i-1}$ declares a subtype T_i of T_{i-1}, where T_{i-1} is a previously declared type or subtype; the interpretation of T_i is then a subset of the interpretation of T_{i-1};
- `p`$:\bar{T} \rightarrow \mathbb{B}$ `(p` $\subseteq$ `q)` states that the interpretation of predicate `p` is a subset of the interpretation of `q`, where `q` is a previously-declared predicate or type;
- `f`$:\bar{T} \rightarrow$`T (domain: q, range: r)` states that the interpretation of function `f` is totally defined over the set interpreting `q`, and its range is a subset of the set interpreting `r`.

This syntax is further detailed in Appendix A on page 119. An example follows.

Example 9. *The predicate* `marriedPerson: Person → 𝔹` *is interpreted as a subset of* `Person`*. The declaration of function* `spouseOf` *is :*

```
spouseOf : Person→Person (domain: marriedPerson,
                          range:marriedPerson)
```

Although the interpretation `spouseOf` *is partial over the interpretation of* `Person`*, it is total over the interpretation of* `marriedPerson`*. The sentence* `marriedPerson(Tom) ∧ spouseOf(Tom)=Mary` *is non-ambiguous. In the sentence with binary quantification* `∃ p ∈ marriedPerson: spouseOf(p)=Mary`*,* `p` *ranges over the interpretation of* `marriedPerson`*, i.e., over the domain of* `spouseOf`*, and the term* `spouseOf(p)` *is provably denoting in its context.*

The domain of partial functions may also be (the cross-product of) constructed types, as the following example shows:

Example 10. *Consider a crossword: one of its cell is identified by a row and a column number; only a white cell may contain a letter. A partially-filled crossword can be represented with this vocabulary:*

Listing 2.1: Vocabulary for crosswords

```
vocabulary V {
  type Row, Column, Letter
  type Cell ≜ constructed from {cell(col:Column,row:Row)}
  isWhite :  Cell →  𝔹
  hasLetter: Cell →  𝔹  (hasLetter ⊆ isWhite)
  letter:    Cell → Letter (domain: hasLetter)
}
```

Notice that `letter` *is a partial function over the grid of the crossword: it is defined on white cells that have a letter.*

Markovic et al. (2023) also proposes two additional language constructs to reduce verbosity. In their proposal, sentences, as well as each quantified formula in them, must be annotated with *implicit guards*. The annotation $[\![\phi]\!]$ stands for *implicit conjunctive guards* and the annotation $\langle\langle\phi\rangle\rangle$ for *implicit implicative guards*. These guards disambiguating the two meanings proposed by Bertrand Russell for sentences with potentially non-denoting terms. When a formula is implicitly guarded, it is extended with guards for all the terms and sub-terms it contains. For example, formula $\langle\langle isAlive(spouseOf(Tom))\rangle\rangle$ stands for $\delta_{Tom} \wedge \delta_{spouseOf}(Tom) \Rightarrow isAlive(spouseOf(Tom))$, while formula $[\![isAlive(spouseOf(Tom))]\!]$ stands for $\delta_{Tom} \wedge \delta_{spouseOf}(Tom) \wedge isAlive(spouseOf(Tom))$.

These implicit guards are not part of traditional mathematical notations, and are not currently included in the specification of FO[·]. They may be added later when practice shows that knowledge bases are too verbose.

`Concept` as a type Sometimes, it is necessary to reason about the concepts in an ontology, as the following example shows.

Example 11. *Consider a corona testing protocol. A person is to be tested if she has at least two of the following symptoms: fever, coughing and sneezing. Following KR best practices, the set of persons showing a symptom is represented by a unary predicate ranging over persons, i.e., fever*/1*, coughing*/1*, and sneezing*/1*. Leaving first-order logic behind us, assume that the set of symptoms is represented by the predicate symptom*/1*.*

Testing is expressed as a unary predicate test1/1*, ranging over persons. Informally, this predicate is to be defined along the following lines (while freely extending the syntax of FO):*

$$\forall p(test1(p) \Leftrightarrow 2 \leqslant \#\{s|symptom(s) \wedge s(p)\}) \tag{2.1}$$

We argued in Carbonnelle et al. (2023b) that the variable s in the cardinality aggregate above is neither a second-order variable ranging over the set of sets of person, nor a variable ranging over the set of symbols in the vocabulary, but a variable ranging over the set of concepts in the category of symptoms. Informally, the *concept* behind a symbol is its informal interpretation.

In classical first-order logic, a structure only has the *extension* of symbols. The extension of the symbol σ in a structure is its *formal* interpretation, also known as its value, in that structure. For our purpose, we extend a structure to also have the *intension* of symbols, i.e., an element in the domain of discourse representing their *informal* interpretation.

In philosophical logic, the concepts of *intension* and *extension* have been investigated at least since Frege (for an historical overview, see Fitting, 2020). A prototypical example concerns the "morning star" and "evening star", i.e., the star visible in the east around sunrise, and in the west around sunset. These words have two different intensions (i.e., they refer to two different informal interpretations, i.e., two different concepts), but as it happens, in the current state of affairs, their extensions (i.e., their formal interpretation) are the same: the planet also known as Venus.

In intensional logic, the intension of an expression is modelled as a mapping from possible worlds to the interpretation of the expression. For example, the intension of *EveningStar* is the mapping from possible worlds to the extension

of *EveningStar* in that world. Furthermore, possible worlds are linked by an *accessibility* relation. Modal operators, e.g., $\Diamond, \Box$, are used to talk about the different values that expressions may have in the *current* and in *accessible* worlds.

In FO(·), we eschew these modal operators but introduce quantifications and aggregations over concepts. The intension of the symbol σ is denoted by the identifier ‘σ (with a back-tick). The type of an intension is called `Concept`. The interpretation of type `Concept` is the set of intension of the symbols in the vocabulary. For convenience, the intensions of symbols having the same type signature are grouped into subtypes of `Concept`, denoted using square brackets: for example, `Concept[Person` $\rightarrow \mathbb{B}$`]` is the set of intensions having the signature `Person` $\rightarrow \mathbb{B}$. Finally, FO(·) introduces the *value functor* $eval(\cdot)$: when applied to an intension, it returns its formal interpretation in the current structure. Thus, `$(‘hasFever)(p)` has the same value as `hasFever(p)`.

Example 12. *The number of symptoms for a person* `p` *can now be formulated as* `#{s∈symptoms: $(s)(p)}` *where* `symptoms≜{‘hasFever,‘sneezes, ‘coughs}` *is the set of symptom concepts. The type signature of* `symptoms` *is* `Concept[Person` $\rightarrow \mathbb{B}$`]` $\rightarrow \mathbb{B}$.

Unlike meta-programming in logic programming language (such as Prolog (Barklund et al., 2000; Hill & Gallagher, 1998), HiLog (Chen et al., 1993) or webdamlog (Abiteboul et al., 2011)), FO(·) enforces the requirement that sentences be well-typed. For example, `$(‘hasFever)` must be applied to exactly one argument of type Person; otherwise an error is raised. This helps the developer of a knowledge base avoid typical encoding errors. The method to enforce this requirement is similar to the one used to enforce the well-definedness of sentences in the presence of partial functions, and is described in Appendix A on page 119.

An alternative approach to intensional objects would be to “reify” the symptoms, i.e., to introduce a `type Symptom≜{Fever,Sneezing,Coughing}`, and a predicate `has: Symptom` $\times$ `Person` $\rightarrow \mathbb{B}$ to represent the relation between symptoms and persons. This approach is not elaboration tolerant (McCarthy, 1998), though: it would require changing the theory to use reified concepts when aggregates over concepts must be introduced. Elaboration tolerance is a desirable feature of a KR formalism, as it helps reduce development costs.

2.3 IDP-Z3, a reasoning engine for FO[.]

We have developed a reference implementation for the proposed FO[·] standard, called IDP-Z3[2]. IDP-Z3 can be run from the command line, or integrated in a Python application as a downloadable Python package.[3] Computations can also be run online via a webIDE[4]. It is open source[5] under the LGPL 3 license, and its documentation is available online[6].

The development priority was to make it feature-complete and correct; some efforts have also been put to have adequate performance. It has a test suite of 200+ files, and we use a formal process to discuss enhancements (using IDP Enhancement Proposal, IEP).[7]

IDP-Z3 can solve a standard model expansion (or model optimization) problem expressed in FO[·]. It can also perform many other reasoning tasks using the knowledge expressed in FO[·], as we'll see in the next chapter (Section 3.4 on page 49).

Example 13. *Listing 2.2 is the full knowledge base for the Railway problem.*

Listing 2.2: Knowledge base for the Railway problem

```
vocabulary V {
    type City ≜ {A, B, C}
    cost : City × City → Int
    railtrack, reachable: City × City → 𝔹
    totalCost: () → Int

    var c in City
}
structure S:V {
    cost ≜ { (A,A)→0, (B,B)→0, (C,C)→ 0,
             (A,B)→13, (B,A)→13,
             (B,C)→45, (C,B)→45,
             (A,C)→52, (C,A)→52 }.
}
theory T:V {
 {∀ c: reachable(c,c).
  ∀c1,c2:reachable(c1,c2)← railtrack(c1,c2)∨railtrack(c2,c1)
  ∀ c1,c2: reachable(c1,c2) ←
```

[2] https:www.idp-z3.be
[3] https://pypi.org/project/idp-engine/
[4] https://interactive-consultant.idp-z3.be/IDE
[5] https://gitlab.com/krr/IDP-Z3
[6] https://docs.idp-z3.be
[7] https://gitlab.com/krr/IDP-Z3/-/wikis/Home

```
        ∃ c3: reachable(c1,c3) ∧ reachable(c3,c2).
 }
 ∀ c1,c2: reachable(c1,c2).
 totalCost()=sum{{cost(c1,c2) | (c1, c2) in railtrack}}.
}
procedure main() {
    pretty_print(minimize(T, S, term="totalCost()"))
  }
```

The solution that minimizes `totalCost()` *is:*

Listing 2.3: Solution of the Railway problem

```
  cost ≜ {(A, A) → 0, (A, B) → 13, ...}.
  railtrack ≜ {(B, A), (B, C)}.
  reachable ≜ {(A, A), (A, B), (A, C),
               (B, A), (B, B), (B, C),
               (C, A), (C, B), (C, C)}.
  totalCost ≜ 58.
```

IDP-Z3 is the successor of IDP3 (De Cat et al., 2013), which used a custom SAT solver, called minisat(ID) (De Cat et al., 2014). Hence, IDP3's support for arithmetic was limited. By contrast, IDP-Z3 uses an off-the-shelf SMT solver, called Z3 (De Moura & Bjørner, 2008), which supports non-linear arithmetic reasoning over the set of rationals. This significantly increases the field of applications since, for example, IDP-Z3 can handle the value of physical measurements.

A knowledge base is a set of vocabularies and theories in a Unicode text file. Most connectives can be entered using Unicode or ASCII characters (e.g. $\wedge$ or `&` for a conjunction).

A model expansion problem $\mathcal{MX}(\Sigma, I_{env}, \phi)$ consists of a vocabulary Σ, a theory ϕ and a data theory I_{env}. The theories are translated into a formula acceptable by Z3 in multiple steps:

(i) the text file is parsed;

(ii) the theories are annotated with type information and their well-formedness is verified;

(iii) inductive definitions are reduced to first-order logic using level mapping;

(iv) the enumeration for predicate and function symbols are translated into atoms;

Original formula	Translation
$\forall \bar{x} \in p : \phi(\bar{x}, \bar{y})$	$\forall \bar{x} \in \bar{T} : \neg p(x) \vee \phi(\bar{x}, \bar{y})$
$\exists \bar{x} \in p : \phi(\bar{x}, \bar{y})$	$\exists \bar{x} \in \bar{T} : p(x) \wedge \phi(\bar{x}, \bar{y})$
$\#\{\bar{x} \in p \mid \phi(\bar{x}, \bar{y})\}$	$\#\{\bar{x} \in \bar{T} \mid p(x) \wedge \phi(\bar{x}, \bar{y})\}$
$sum\{\{f(\bar{x}, \bar{y}) \mid \bar{x} \in p : \phi(\bar{x}, \bar{y})\}\}$	$sum\{\{f(\bar{x}, \bar{y}) \mid \bar{x} \in \bar{T} : p(x) \wedge \phi(\bar{x}, \bar{y})\}\}$
$min\{f(\bar{x}, \bar{y}) \mid \bar{x} \in p : \phi(\bar{x}, \bar{y})\}$	$min\{f(\bar{x}, \bar{y}) \mid \bar{x} \in \bar{T} : p(x) \wedge \phi(\bar{x}, \bar{y})\}$
$max\{f(\bar{x}, \bar{y}) \mid \bar{x} \in p : \phi(\bar{x}, \bar{y})\}$	$max\{f(\bar{x}, \bar{y}) \mid \bar{x} \in \bar{T} : p(x) \wedge \phi(\bar{x}, \bar{y})\}$

Table 2.3: Resolution of binary quantifications and aggregates for $p : \bar{T} \to \mathbb{B}$

(v) the enumerations of types are used to instantiate (i.e., ground) the quantifications and aggregates;

(vi) the ground formulas are submitted to Z3 for satisfiability solving.

A challenge in migrating IDP3 to the Z3 SMT solver was in re-implementing some specific capabilities of minisat(ID). In particular, minisat(ID) used custom procedures to handle (inductive) definitions. We now describe some technical aspects of the IDP-Z3 implementation.

Grounding

The grounding algorithm has two goals: replace quantifications and aggregations over finite sets by discrete formulas, and simplify formulas by injecting the values listed in enumerations. For example, $\forall x \in T : p(x)$ is grounded to $p(1) \wedge p(2)$ when T is interpreted as the set $\{1, 2\}$. It further reduces to true if the set interpreting p is also $\{1, 2\}$.

Quantification and aggregates may range over a set interpreting a predicate, in so-called binary quantification and aggregates, e.g., `∃(c1,c2) ∈ railtrack: c1=A`. Such quantifications and aggregates are reduced to quantification and aggregates over types according to Table 2.3.

Quantifications and aggregates over finite types $\bar{T}$ and with free variables $\bar{y}$ are translated according to Table 2.4. By notation, when $\bar{T}$ is simply T, its interpretation is of the form $\{t_1, \ldots, t_n\}$; when $\bar{T}$ is a tuple, its interpretation is of the form $\{\bar{t}_1, \ldots, \bar{t}_n\}$, where each $\bar{t}_i$ is a tuple in the cross-product of the interpretation of each T_i. Note that the second additional sentence for the minimum (and maximum) aggregate (see notes in the Table) requires that $\phi(\bar{x}, \bar{y})$ is satisfiable for any $\bar{y}$, i.e., that the set to minimize over is not empty for any $\bar{y}$.

Finally, when grounding a quantification, as soon as an instantiation of a predicate or function application results in a ground expression, and when the

Original term	Translation
$\forall \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y})$	$\phi(\bar{t}_1, \bar{y}) \wedge \cdots \wedge \phi(\bar{t}_n, \bar{y})$
$\exists \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y})$	$\phi(\bar{t}_1, \bar{y}) \vee \cdots \vee \phi(\bar{t}_n, \bar{y})$
$\#\{\bar{x} \in \bar{T} \mid \phi(\bar{x}, \bar{y})\}$	**if** $\phi(\bar{t}_1, \bar{y})$ **then** 1 **else** 0 $+$**if** $\phi(\bar{t}_2, \bar{y})$ **then** 1 **else** 0 $\cdots +$ **if** $\phi(\bar{t}_n, \bar{y})$ **then** 1 **else** 0
$sum\{\{f(\bar{x}, \bar{y}) \mid \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y})\}\}$	**if** $\phi(\bar{t}_1, \bar{y})$ **then** $f(\bar{t}_1, \bar{y})$ **else** 0 $+$**if** $\phi(\bar{t}_2, \bar{y})$ **then** $f(\bar{t}_2, \bar{y})$ **else** 0 $\cdots +$ **if** $\phi(\bar{t}_n, \bar{y})$ **then** $f(\bar{t}_n, \bar{y})$ **else** 0
$min\{f(\bar{x}, \bar{y}) \mid \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y})\}$	$min_f(\bar{y})$ (*)
$max\{f(\bar{x}, \bar{y}) \mid \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y})\}$	$max_f(\bar{y})$ (**)

(*) min_f is a new function symbol, and the following sentences are added to the theory: $\forall \bar{y} \in \bar{T}_y : \forall \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y}) \Rightarrow min_f(\bar{y}) \leqslant f(\bar{x}, \bar{y})$ and $\forall \bar{y} \in \bar{T}_y : \exists \bar{x} \in \bar{T} : \phi(\bar{x}, \bar{y}) \wedge min_f(\bar{y}) = f(\bar{x}, \bar{y})$.
(**) Similar to min.

Table 2.4: Grounding of quantifications and aggregates

predicate or function symbol has an enumeration, the ground expression is replaced by its value given by the enumeration. This value is propagated when possible: if the value is **true** and is part of a disjunction, the disjunction is necessarily **true**, and the rest of the disjunction does not have to be grounded. The same can be said of **false** in a conjunction. Such immediate simplification improves the speed of grounding.

Quantification over infinite domains cannot be grounded: they are transmitted without changes to Z3. Minimum and maximum aggregates over infinite domain are also transmitted to Z3 because they reduce to quantifications over infinite domains. Z3 can sometimes find models of such formulas, but not always. Indeed, satisfiability checking over infinite domains is undecidable. Sum aggregates over infinite domains raise an error because they cannot be handled by Z3.

Non-inductive definitions are translated to formulas acceptable by Z3 using Clark's completion semantics (Clark, 1977): a defined atom must have the same truth value as the disjunction of the body of the rules defining it. The following listing shows a non-inductive definition and its translation using Clark's completion:

Listing 2.4: Connected relation

```
{ ∀ c1,c2: connected(c1,c2)←railtrack(c1,c2).
  ∀ c1,c2: connected(c1,c2)←railtrack(c2,c1).
}
```

```
// Clark's completion:
∀ c1, c2: connected(c1,c2)
     ⇔  railtrack(c1,c1) ∨ railtrack(c2,c1).
```

Inductive definitions of predicates are reduced to formulas acceptable by Z3 using a mechanism proposed by Pelov to implement the well-founded semantics of definitions (Pelov & Ternovska, 2005). A *level-mapping* function symbol is added to the vocabulary for every inductively defined predicate symbol. The interpretation of this function associates a number to each tuple of values in the domain of the defined predicate. These numbers define the induction order[8]. Inequalities on level-mappings are included in the translation of the definition to ensure that this induction order is safe in a model of the definition.

For illustrative purposes, we consider the definition `{ p ← q. q ← p.}`. The Clark completion of the definition above is `p ⇔ q. q ⇔ p.` It has a spurious model where both `p` and `q` are true. To eliminate this spurious model, the translation of the definition proposed by Pelov & Ternovska (2005) adds level mapping constraints to the body of rules, as follows:

Listing 2.5: Level mapping for a simple definition

```
 p ⇒ ( q ∧ step_p >  step_q).
¬p ⇒ (¬q ∧ step_p ⩾ step_q).
 q ⇒ ( p ∧ step_q >  step_p).
¬q ⇒ (¬p ∧ step_q ⩾ step_p).
```

`step_p` (resp. `step_q`) represents the index of the iteration in the inductive process at which the truth value of `p` (resp. `q`) is determined. The first line informally reads as follows: the necessary condition for determining that `p` is true is that `q` has been determined to be true strictly before `p`. The second line states that the necessary condition for determining that `p` is false is that `q` was determined to be false before or at the same time as `p`, thus allowing for `p` and `q` to be determined as false at the start of the induction process. The translated sentences have only one model, where both `p` and `q` are false, unlike the Clark completion of the definition.

These conditions are derived from the construction mechanism described in Section 1.3 on page 6. The mechanism also generalizes to non-stratified definitions (by excluding models whose justification has "infinite mixed loops"), but that is beyond the scope of this thesis.

[8] In practice, the co-domain of the level mapping function is the set of rational numbers (and not the set of natural numbers): it gives Z3 more freedom to choose level-mapping numbers, resulting in better performance.

The well-founded semantics described above is used by default in IDP-Z3 because it matches with how humans intuitively understand inductive definitions (Denecker & Vennekens, 2014). Still, for training purposes, a definition can be specified to use the completion, co-inductive, stable, or Kripke-Kleene semantics instead. Variations of the inequalities above are used to support these semantics, as illustrated in Tables 2.5 and 2.6 on the next page. The level mappings for the stable semantics have been proposed by Janhunen (2004). Support for the Founded and Constraint semantics may be provided in the future (Liu & Stoller, 2020, 2022).

Clark completion	$P \Leftrightarrow Q.$ $Q \Leftrightarrow P.$
Stable semantics	$P \Leftrightarrow (Q \wedge level_P > level_Q).$ $Q \Leftrightarrow (P \wedge level_Q > level_P).$
Well-founded semantics	$P \Rightarrow (Q \wedge level_P > level_Q).$ $\neg P \Rightarrow (\neg Q \wedge level_P >= level_Q).$ $Q \Rightarrow (P \wedge level_Q > level_P).$ $\neg Q \Rightarrow (\neg P \wedge level_Q >= level_P).$
Co-inductive semantics	$P \Rightarrow (Q \wedge level_P \geqslant level_Q).$ $\neg P \Rightarrow (\neg Q \wedge level_P > level_Q).$ $Q \Rightarrow (P \wedge level_Q \geqslant level_P).$ $\neg Q \Rightarrow (\neg P \wedge level_Q > level_P).$
Kripke-Kleene semantics	$P \Rightarrow (Q \wedge level_P > level_Q).$ $\neg P \Rightarrow (\neg Q \wedge level_P > level_Q).$ $Q \Rightarrow (P \wedge level_Q > level_P).$ $\neg Q \Rightarrow (\neg P \wedge level_Q > level_P).$

Table 2.5: Translation of `{ P ← Q. Q ← P.}`

An important aspect of the model expansion of definitions is that the structure I_{env} in the $\mathcal{MX}(\Sigma, I_{env}, \phi)$ problem can be any partial structure. As an example, consider the case where formula ϕ is the definition of the reachability relation (Listing 1.1 on page 7). In a first problem, the interpretation of the `railtrack` relation is given in I_{env}; model expansion yields an interpretation of the `reachable` relation. In a second problem, an interpretation of the `reachable` relation is given instead; model expansion on that same definition will now yield an interpretation of the `railtrack` relation such that its transitive closure is the given `reachable` relation.

By contrast, in logic programming and answer set programming, solving these two distinct problems would require significantly different programs. This example illustrates the difference between programming and knowledge representation.

Clark completion	$P \Leftrightarrow \neg Q.$ $Q \Leftrightarrow \neg P.$
Stable semantics	$P \Leftrightarrow \neg Q.$ $Q \Leftrightarrow \neg P.$
Well-founded semantics	$P \Rightarrow \neg(Q \wedge level_P > level_Q).$ $\neg P \Rightarrow (Q \wedge level_P >= level_Q).$ $Q \Rightarrow \neg(P \wedge level_Q > level_P).$ $\neg Q \Rightarrow (P \wedge level_Q >= level_P).$
Co-inductive semantics	$P \Rightarrow (\neg Q \wedge level_P \geqslant level_Q).$ $\neg P \Rightarrow (Q \wedge level_P > level_Q).$ $Q \Rightarrow (\neg P \wedge level_Q \geqslant level_P).$ $\neg Q \Rightarrow (P \wedge level_Q > level_P).$
Kripke-Kleene semantics	$P \Rightarrow (\neg Q \wedge level_P > level_Q).$ $\neg P \Rightarrow (Q \wedge level_P > level_Q).$ $Q \Rightarrow \neg(P \wedge level_Q > level_P).$ $\neg Q \Rightarrow (P \wedge level_Q > level_P).$

Table 2.6: Translation of `{ P ← ¬Q. Q ← ¬P.}`

Recursive definition of functions IDP-Z3 can reason with recursive definitions of functions, as long as the defined symbol does not appear in compound terms, e.g., in aggregates. Hence, the Company control example (Example 5 on page 9) is not supported.

The Z3 solver also proposes a mechanism to find models of definitions over recursive data types and over integers. Per its documentation,[9] "Z3 unfolds the definition of recursive functions incrementally by iterative deepening: it attempts first to establish satisfiability modulo a fixed bound on the number of unfoldings; if the resulting formula is unsatisfiable and the reason for unsatisfiability is due to the bound restriction, it increases the fixed bound incrementally." IDP-Z3 uses this capability when the definition is annotated with the `(recursive)` keyword. As an example, Listing 2.6 has the definition of the length of a list of integers[10].

Listing 2.6: Length of a list

```
vocabulary V {
  type List ≜ constructed from {nil,cons(car:Int, cdr: List)}
  len: List → Int
}
theory T:V {
```

[9] `https://microsoft.github.io/z3guide/docs/logic/Recursive%20Functions/`

[10] `is_cons` is a function implicitly declared by the declaration of `List`: it is true when its argument is a list constructed with `cons`.

```
  { (recursive)
    len(nil) = 0.
    ∀ x in List: len(x) = len(cdr(x))+1 ← is_cons(x).
  }
}
```

Partial functions The checking of well-definedness of sentences involving partial functions is done according to Appendix A.

The interpretation of a function symbol in a Z3 model is total. IDP-Z3 filters this interpretation to restrict it to the domain of the symbol. Two IDP-Z3 structures over the same vocabulary are considered different only when the interpretations restricted to the domain of the symbols are different. Thus, when enumerating different models of a theory, only models that differ in a significant way are returned.

2.4 Discussion

Tables 2.7 on the following page compare IDP-Z3's capabilities to those of Z3 (an SMT solver), clingo (an ASP solver), and IDP3 (the predecessor of IDP-Z3).

IDP-Z3 is essentially a wrapper around the Z3 SMT solver, adding expressivity and convenience to the standard SMT language, called SMT-Lib2 (Barrett et al., 2010b): in particular, IDP-Z3 adds support for aggregates, inductive definitions, partial functions and intensional objects. It also uses a familiar math syntax. As we'll see in the next chapter, it also adds new reasoning capabilities to Z3, namely the determination of relevant symbols. However, IDP-Z3 adds some overhead to Z3 in terms of performance: this overhead could be reduced in future work. Still, IDP-Z3 is used as a reasoning engine in practical applications, as we'll see in the next chapter.

Compared to its predecessor (IDP3), IDP-Z3 adds support for arithmetic over the infinite set of integers and rationals, opening up a new field of applications. It also adds support for intensional objects.

We mentioned earlier that some problem classes should be "universally solvable" (Section 2.1 on page 11): every interpretation of the input symbols in $\Sigma_{I_{env}}$ should be such that there is an interpretation of the remaining symbols in Σ that satisfy ϕ. Alternatively, there should be no interpretation of the input symbols such that, for every interpretation of the remaining symbols, ϕ is not satisfied.

Language feature	IDP-Z3	Z3	clingo 5	IDP3
Classical FO	✓	✓		✓
Types, quantification	✓	✓		✓
Uninterpreted types		✓		
(Inductive) definitions	✓	*	*	✓
Disjunction in the head			✓	
Linear arithmetic	✓	✓	✓	**
Non-linear arithmetic	✓	✓	**	**
Aggregates	✓		✓	**
Partial function	✓			✓
Vector, Array		✓		
Concept as a type	✓			

(*) Z3 supports recursive definitions of functions (but not inductive definition of predicates); clingo has recursive rules, but see them as implications under stable semantics (instead of definitions under the well-founded semantics).
(**) on finite domain only.

Table 2.7: Comparison of language features of solvers.

Example 14. *Consider the model expansion problem where Σ has two nullary symbols, a and x, whose values are in the interval* 0..3*; a is an environmental variable (its interpretation is given); x is a decision variable; the sentence to be satisfied is that the sum of* `a` *and* `x` *is 3. The problem class is* $(\Sigma, \Sigma_{I_{env}}, \phi)$ *where:*

- Σ *is* `{type T ≜ {0..3}, a,x:() → T}`
- $\Sigma_{I_{env}}$ *is* Σ *without* `x`*, i.e.,* `{type T ≜ {0..3}, a:() → T};`
- ϕ *is* $a() + x() = 3$

This problem class is universally solvable if and only if there is no interpretation of a such that there is no interpretation of x that satisfies the sentence, i.e., iff $\mathcal{MX}(\Sigma_{I_{env}}, \{T := \{0..3\}\}, \neg\exists x \in T : a() + x = 3)$ *has no model interpreting a. Here, x is a variable ranging over the possible interpretation of the term* `x()`*, i.e., over the range* 0..3*. IDP-Z3 can be used to show that this $\mathcal{MX}$ problem has no model given the interpretation of the type* `T`*: thus, the problem class is universally solvable. It would not be universally solvable if the interpretation of the type* `T` *were* `0..2`*, or if the sum of* `a` *and* `x` *had to be 4.*

When the unknown symbols in $\Sigma \backslash \Sigma_{I_{env}}$ are n-ary, universal-solvability cannot be expressed in first-order logic anymore, but can be expressed in the $\forall\exists SO$

fragment of second-order logic (SO). Second-order logic is an extension of first-order logic in which variables can represent not only elements in the domain of discourse, but also n-ary relations and functions. The $\forall\exists SO$ fragment is the set of SO sentences in which the existential quantifiers precede the universal quantifiers. Thus, second-order variables can be used to represent "the interpretation of the remaining symbols". Verifying the universally-solvable property of a problem class can then be reduced to a $\forall\exists SO$ model search problem. Solving it requires a second-order grounder (van der Hallen & Janssens, 2018), and is currently beyond the capability of IDP-Z3.

2.5 Personal contribution

This chapter is based on (Carbonnelle et al., 2022) and (Carbonnelle et al., 2023b). I wrote the FO[·] standard following discussions with Prof. Denecker and Matthias Van der Hallen. Ingmar Dasseville developed a proof of concept for IDP-Z3: I expanded it extensively with some help from Jo Devriendt and Simon Vandevelde. I developed the grounding algorithm, as well as the proof of concept and the implementation of the support of intensional objects and of subtypes for partial functions in IDP-Z3. Matthias Van der Hallen wrote a draft version of the paper on quantification over intensional objects (Carbonnelle et al., 2023b). Some paragraphs on partial functions in section 2.2 on page 16 are adapted from Markovic et al. (2023).

Chapter 3

Interactive Model Expansion

A limitation of the standard model expansion approach is that the environment and the conditions to be satisfied must be fully known by the model expansion algorithm to obtain an appropriate solution. To overcome this limitation, we explain how an interactive system can assist a user in finding an appropriate solution. We present the Knowledge Base Paradigm that accelerates the development of such interactive systems. We present our theoretical contributions, i.e., a formal analysis of the "split model expansion" problem (Carbonnelle et al., 2023c), and we evaluate our implementation, the Interactive Consultant.

3.1 Introduction

The standard model expansion problem, as defined in Definition 1 on page 14, takes as input a partial structure I_{env} describing the environment, and a sentence ϕ describing the conditions to be satisfied. Both the environment and the conditions must be fully described to obtain an appropriate solution using a generic model expansion algorithm. If the environment is not fully described, the generic algorithm will automatically assign values to the uninterpreted environmental symbols, and those values may not correspond to the environment faced by the user of the system: the solution could then be inappropriate for the user. Similarly, if some conditions are not represented in the sentence, the solution returned by the algorithm may not satisfy them.

This full knowledge is not always available, however. The conditions are

sometimes too complex or numerous to justify their formalization in a sentence: this is notoriously the case for the nurse rostering problem, where personal constraints must often guide the scheduling of the nurses: many constraints are not explicitly given to the system (Drake, 2014). Asking the model expansion algorithm to generate tentative solutions until an acceptable solution is found would put too much burden on the user. A better approach is to provide the user with an interactive system that allows him to progressively build a roster acceptable to him, while the system verifies that the conditions that it knows are satisfied.

Full knowledge of the environment may be lacking too. In many cases, instead of full knowledge of the environment, the environment can be observed when needed, to obtain the missing information. When the knowledge of the environment is incomplete, a generic model expansion algorithm would fill all the blanks, i.e., would choose values for all symbols without interpretation, including environmental symbols. For symbols describing the environment, there are two possibilities: (i) either the values proposed by the system are irrelevant, in the sense that changing those values to match the environment would also give a solution, or (ii) some values are relevant, and those values must be verified, by observation, to match the environment to make sure that the solution is appropriate. If that verification is incomplete, the user might incorrectly accept an inappropriate solution. The question is then: "How could a computerized system assist him in making this verification?" We are not aware of any system with this capability.

Consider the case of a notary public preparing the registration of a real estate sale (motivating Example 2, Chapter 1.1.2 on page 2): initially, he does not know many details of the sale, and he queries his client to obtain the relevant information during his conversation with him. Asking an irrelevant question would not be appropriate. Hence, some information about the environment will remain unknown to him, but it should not matter as long as that information is irrelevant. Each time he enters new information in the interactive system during the conversation, the system should (i) tell him which environmental information should be asked to his client because of its relevance, and (ii) alert him when enough environmental information is known to ensure that the solution is appropriate.

To allow the system to determine whether a fact should be verified by observation, it is necessary to distinguish fixed but unknown *environmental* symbols from undecided *decision* symbols. Importantly, next to making a distinction between these two types of symbols, we also make a distinction between the laws of possible environments and the laws of acceptable solutions. The laws of possible environments are guaranteed to hold independently of the decisions of the user; by contrast, the satisfaction of the laws of acceptable solutions depends on

the decision of the user. In this chapter, we analyze the consequences of these distinctions on the knowledge representation and on the reasoning required to solve model expansion problems interactively.

Unlike consequences of the laws of possible environments, environmental prerequisites derived from the laws of acceptable solutions *must be* verified by observation. To see this, consider a real estate sales tax law with assertions stating which county belongs to which states, and what conditions a buyer must meet to obtain a tax rebate in each state. The first assertions are part of the laws of possible environments, while the second assertions are part of the laws of acceptable solutions. If the property is known to be in a county, it is safe to conclude from the first set of assertions that the property is also in the state of that county: it is not necessary to observe the state in which it is. By contrast, suppose that the buyer tentatively considers the option to claim a tax rebate available in only one state. By the second set of assertions, the system concludes that the property must be in that state, but this is not a safe conclusion. Indeed, it is clear that this conclusion must be verified by observing the environment (i.e., by asking the client), and if the property is observed to be in another state, the user must be informed that they cannot claim that tax rebate.

Recall that we consider only problems without uncertainty (Section 1.1 on page 1): the satisfaction of the given conditions of the problem only depends on the fixed environment and on the decisions that are freely made. Hence, we assume that all environmental variables can be observed.

In the next section, we present the state of the art for interactive model expansion. In a later section, we shall explain our approach and discuss its implementation.

3.2 State of the art: the KB paradigm for product configuration

Configuration problems, i.e., the problems of configuring a complex system out of simpler components to satisfy customer requirements, as illustrated by the Railway network problem (Section 1.1.1 on page 2), are important examples of model expansion problems. They have been studied by, e.g., Felfernig et al. (2014). Several authors have discussed interactive tools to solve configuration problems (Madsen, 2003; Van Hertum et al., 2017; Falkner et al., 2019; Schneeweiss & Hofstedt, 2011; Hadzic & Andersen, 2004) when the set of conditions to be satisfied is not fully given to the generic algorithm. Our

theoretical contribution is orthogonal to these studies. Indeed, the explicit distinction we make between environmental and decision symbols, and between the theory of the environment and the theory of acceptable solutions, can be combined with the ideas from earlier work.

Prior work shows that an interactive configuration system needs to provide many functionalities to its user:

- asking him relevant questions,
- computing the consequences of choices made by the user,
- providing explanation for these consequences,
- handling missing information,
- backtracking if the user regrets some choices,
- completing a partial configuration to minimize a cost function,
- checking the validity of a full configuration,
- etc.

Building interactive systems with all these functionalities is renowned for being difficult, and no method to build them is broadly accepted (Felfernig et al., 2014; Axling & Haridi, 1996). The difficulty is compounded when the system offers flexibility in the sequence of data entry, as is often needed to support different types of users. Indeed, different users use different approaches to solve configuration problems: for example, novice users prefer to enter their needs, while expert users prefer to enter their design choices (Randall et al., 2005). This requires the capacity to reason with the particular set of information that a user has given at each step of his search.

Hence, using standard imperative programming is often a nightmare (Barker & O'Connor, 1989; Piller et al., 2014) because (1) many functionalities need to be provided, (2) they are complex to implement, and (3) the resulting programs turn out to be extremely difficult to maintain when existing constraints of the application domain change or new constraints emerge. Indeed, the change of one constraint may require changes in many different (and often non-obviously related) parts of the program code.

For such interactive model expansion, as for the standard model expansion problem, it is natural to look for *declarative solutions*. The research question is: "how can we build interactive configuration systems in which the knowledge of the application domain can be expressed in a formal modelling language, such

that a generic inference system can provide the aforementioned functionalities by performing the required form of reasoning on the knowledge base?"

These challenges make interactive configuration an ideal application domain for the *Knowledge Base Paradigm* (Denecker & Vennekens, 2008). This paradigm returns to the classical view of logic as a formal language to express *pure knowledge* in a declarative way. In this paradigm, an application is developed by writing a *declarative knowledge base* (KB), and by using a knowledge base system that can perform various reasoning tasks using the said KB, by applying different *generic inferences*. Phrased in the terminology of McCarthy (1989), the knowledge in the knowledge base has a *neutrality of purpose, like any human knowledge*, and the knowledge base system is at McCarthy's *fourth level of the scale of intelligent logic machine*. Machines at the fourth level go beyond deductive reasoning and theorem proving and support many other forms of reasoning.

The KB paradigm was followed by Van Hertum et al. (2017) in a prototypical configurator called AutoConfig. This configurator allowed the user to enter data in any order, and, after any data entry, performed various computations to assist the user in his exploration of the solution space. To represent complex knowledge, it used the FO(·) language described earlier. To perform the computations, it used the IDP3 reasoning engine (De Cat et al., 2013). The user interface was generated automatically from the knowledge base: no programming was required, significantly reducing development costs (Deryck et al., 2019).

Our implementation (Carbonnelle et al., 2019) presented further below builds upon AutoConfig and IDP3.

Several declarative approaches for controlling the relevance of symbols in configuration problems have been proposed in the past. In these approaches, the modeler has the responsibility to write so-called *activation rules* that explicitly specify when a symbol becomes active (as done by Mittal & Falkenhainer, 1990; Bowen & Bahler, 1991, in Dynamic CSPs), or even when an entire component becomes active (as done by Stumptner & Haselböck, 1993, in Generative CSPs). Our approach does not need activation rules to derive relevance of symbols. Jansen et al. (2016) also propose an approach without activation rules, based on justification theory. Its implementation requires a custom solver (Deryck et al., 2019). By contrast, our implementation uses a standard SMT solver.

3.3 Theoretical contribution

3.3.1 Split model expansion

We first consider the generic problem of (non-interactive) model expansion in a two-theory setting, using two vocabularies, when the environment is fully known. The first vocabulary, Σ_{env}, contains the *environmental symbols*. The particular environment faced by the user is described by a total structure I_{env} for Σ_{env}. The second vocabulary, Σ_{sol}, is disjoint from the first and contains the decision symbols. A solution is fully described by a total structure I_{sol} for Σ_{sol}. The union of the 2 vocabularies is denoted by Σ_{all}.

We assume to have a theory T_{env} over vocabulary Σ_{env} that contains the laws satisfied by any environment that could be faced by the user. Any structure I_{env} that describes a particular environment is a model of T_{env}. We also assume that the constraints on the solution to the problem faced by the user are formalized by a logic theory T_{sol} in the vocabulary Σ_{all}. This theory fully captures all knowledge about when a solution is acceptable in a given environment.

Definition 3. *The* split model expansion problem $\mathcal{SMX}$ *is a problem that takes as input a tuple* $(\Sigma_{env}, \Sigma_{sol}, I_{env}, T_{env}, T_{sol})$ *where* I_{env} *is a total structure over* Σ_{env}, T_{env} *is a theory over* Σ_{env} *satisfied by* I_{env}, *and* T_{sol} *is a theory over* $\Sigma_{env} \cup \Sigma_{sol}$.
A total solution *for such a problem is a total structure* I_{sol} *over* Σ_{sol} *such that*

$$[\![T_{sol}]\!]^{I_{env}+I_{sol}} = \textbf{\textit{true}}.$$

This definition requires that I_{env} be fully known to verify that I_{sol} is a solution. We will relax this requirement in Section 3.3.4 on page 45. Note that T_{env} is not used in the definition of the total solution because we assume that the environment satisfies T_{env} already. Having T_{env} as an additional input is useful in the process of finding a solution when the environment I_{env} is not fully known, but observable, as discussed in the next sections.

Example 15. *We consider the second motivating example described in Section 1.1.2 on page 2. The two vocabularies and the two theories are shown in Listing 3.1.*

Listing 3.1: Split knowledge base for Real Estate Sale Tax

```
vocabulary environment {
    social_habitat  : () → 𝔹
    licensed_seller : () → 𝔹
    low_rent        : () → 𝔹
```

```
}
vocabulary decision {
    type Registration_Type ≜ {Social, Modest, Other}
    type Tax_Rate ≜ {1, 7, 10}

    tax_rate : () → Tax_Rate
    registration_Type : () → Registration_Type
}

theory environment:environment {
    low_rent() ⇐ social_habitat().
}
theory decision:decision {
    { tax_rate() = 1  ← registration_Type() = Social.
      tax_rate() = 7  ← registration_Type() = Modest.
      tax_rate() = 10 ← registration_Type() = Other.
    }
    licensed_seller() ∧ social_habitat() ⇐
            registration_Type() = Social.
    low_rent() ⇐ registration_Type() = Modest .
}
```

Note that `a ⇐ b` *is read as "a is a necessary condition for b".*

A possible sale is described by

$$I_{env}=\{social_habitat=\textbf{\textit{true}}, licensed_seller=\textbf{\textit{true}}, low_rent=\textbf{\textit{true}}\}$$

The split model expansion problem for this I_{env} *has 3 total solutions:*

$$\{tax_rate=1, registration_Type=Social\},$$

$$\{tax_rate=7, registration_Type=Modest.\},$$

$$and\ \{tax_rate=10, registration_Type=Other.\}.$$

The split model optimization problem to minimize the tax rate has one solution, the one with the 1% tax rate. An interactive assistant for this theory is available online.[1]

3.3.2 Interactive model expansion

Often, a problem has many acceptable solutions in a given environment. Instead of finding a particular solution by automated model expansion, it is preferable

[1] `https://tinyurl.com/simpleRegistration`

that the automated system allows the user to explore the set of acceptable solutions. For example, a property tax legislation may allow the buyer to register the sales in different ways, as in Example 15 on page 40, and the system should allow them to see the various prerequisites and consequences of each legal options, e.g., on the tax rate.

Such systems are also useful when the environment, while fixed, is not fully known by the user at the start of the search, but can be observed as needed: often, full observation of the environment is not required. In those cases, the interactive system can help the user make relevant observations.

We call the two interacting agents `System` and `User`. `System` offers `User` the freedom to choose values for environmental and decision symbols while verifying the validity of the solution that `User` constructs with respect to the 2 theories, T_{env} and T_{sol}. Since a system is often designed for many users, and each user wants a solution for their own environment, we assume it is the task of `User` to communicate information about their environment to `System`. `User` may have additional knowledge that is unknown to `System`. Indeed, it is often the case that some tacit knowledge is not formalized in the theories, but well known by the user. Obviously, `System` cannot use this tacit knowledge; instead, `User` must use it in the appropriate way.

At each step of an interactive process, `User` has constructed two partial structures, I_{obs} and I_{dec}, assigning values to environmental and decision symbols respectively. Those values correspond to observations and to (tentative) decisions respectively.

Definition 4. *A* state of the interactive process *is a pair of partial structures, (I_{obs}, I_{dec}), such that I_{obs} (resp. I_{dec}) is a partial structure over Σ_{env} (resp. Σ_{sol}).*

By contrast, in prior work where there is no distinction between environmental and decision symbols, the state is described by only one partial structure.

Although `System` only has the information that `User` has provided to it, it also knows T_{env} and T_{sol}; hence, it can verify that the state is consistent with T_{env} and T_{sol}:

Definition 5. *A state (I_{obs}, I_{dec}) is* consistent *(with T_{env} and T_{sol}) iff there is an expansion of the state that satisfies both theories:*

$$\exists I \in \widetilde{I_{obs} + I_{dec}} : [\![T_{env} \cup T_{sol}]\!]^I = \textbf{\textit{true}}$$

During the interactive process, decisions can be added and retracted. We use $I_1 <_p I_2$ to denote that I_1 is strictly less precise than I_2. This leads to the following definition:

Definition 6. *An* interactive process *for a problem $\mathcal{SMX}$ with input $(\Sigma_{env}, \Sigma_{sol}, I_{env}, T_{env}, T_{sol})$ is a sequence $(I^i_{obs}, I^i_{dec})_{0<i\leqslant n}$ of n consistent states, such that, at each step, either the set of observations or the set of decisions is made either more precise or less precise, i.e., for any $0 < i < n$:*

- $(I^i_{obs} <_p I^{i+1}_{obs} \vee I^{i+1}_{obs} <_p I^i_{obs}) \wedge I^i_{dec} = I^{i+1}_{dec}$
- *or,* $I^i_{obs} = I^{i+1}_{obs} \wedge (I^i_{dec} <_p I^{i+1}_{dec} \vee I^{i+1}_{dec} <_p I^i_{dec})$.

In the interactive process, `User` and `System` work together towards a solution as follows:

- The interaction starts with a pair of empty structures, (I_{obs}, I_{dec}). If this state is inconsistent with T_{env} and T_{sol}, `System` tells `User` that there is no solution to the split model expansion problem.
- `User` can make I_{obs} or I_{dec} more precise as long as the new state is consistent. `System` prevents `User` from changing their decisions I_{dec} to an inconsistent state because it would not bring them closer to a solution. If `User` needs to make I_{obs} more precise following an observation, and if this change makes the state inconsistent, `System` asks `User` to retract a decision first so that the new observation can then be added without making the state inconsistent. To facilitate this task, `System` shows the list of decisions that, if retracted, would allow the entry. This list is the list of interpretations in the $\leqslant_p$-minimal structure I less precise than I_{dec} that, together with I_{obs}, is inconsistent with $T_{env} \bigcup T_{sol} \bigcup O$, where O is the observation to be added, in the form of a sentence. It can be obtained using unsat_core methods (Cimatti et al., 2007). By repeatedly making the state more precise while keeping it consistent, `User` eventually obtains a total solution.
- `User` can make I_{obs} or I_{dec} less precise, i.e., retract past observations and decisions. The resulting state is necessarily consistent. This allows `User` to explore the complete space of solutions in their environment.

Example 16. *Returning to Example 15 on page 40, assume that* `User` *first decides that Registration_Type=Social. The corresponding state* ({}, {*Registration_Type=Social*}) *is consistent. They proceed by observing* $\neg$*social_habitat. This cannot be added to the state because social_habitat is a prerequisite for a Social registration, per T_{sol}.* `User` *thus needs to retract their only decision (Registration_Type=Social) before entering their observation, resulting in the state* ({*social_habitat =* ***false***}, {}).

Suppose instead that `User` *selects Registration_Type =Social followed by low_rent =* ***false****. The resulting state is consistent with T_{sol} in the absence of*

T_{env}. However, the state is inconsistent with T_{env} because a Social registration can only be done for a social housing, which is necessarily leased at a low rent. `User` *must backtrack on their decision Registration_Type=Social before entering the observation that the housing is not leased at a low rent.*

Note that the presence of T_{env} reduces the number of consistent states, and thus shortens the search for a solution.

3.3.3 Propagation

`System` further assists `User` by informing them of the logical consequences of what is known in the current state. Such knowledge is of great value to `User`, and is used in most interactive assistant tools (Van Hertum et al., 2017; Falkner et al., 2019). It reduces the number of decisions that `User` has to make, and shortens the interaction in a way analogous to how unit propagation accelerates the naive backtracking algorithm in propositional satisfiability checking. The consequences of what is known are determined by a computation called *propagation* (or *backbone* computation).

Definition 7. *A partial structure I' is a* T-propagation *of a partial structure I iff it is an expansion of I with the same set of model expansions for T as I, i.e.,*

- $I \leqslant_p I'$
- $\forall J \in \overline{I} : (\llbracket T \rrbracket^J = \boldsymbol{true}) \Rightarrow J \in \overline{I'}$

Structure I' expands I by selecting some of the uninterpreted symbols whose value is the same in all models of T that expand I, and assigning them that value. We call I' the *optimal propagation* of I with respect to T if it is the most precise T-propagation. Algorithms to compute propagation are discussed in the literature (e.g., Zhang et al., 2019; Bjørner et al., 2018).

In our interactive assistant framework, a complication arises when propagation derives facts for environmental symbols using the theory of solutions. In general, there is no guarantee that these facts are true in the environment faced by the user. To address this issue, we first propagate using T_{env} only, and then using both T_{env} and T_{sol}. This will further demonstrate the benefits of the split theory setting.

Because T_{env} only concerns environmental symbols, the propagation of the state by T_{env} can assign new values only to environmental symbols. We denote the structure containing these new values by I^e_{obs}. The propagation of the state by

both T_{env} and T_{sol}, i.e., by T_{all}, can assign new values to both environmental and decision symbols. We denote the structures containing these new values by I^a_{obs} and I^a_{dec}. `System` displays these structures differently:

- I^e_{obs}: The new environmental facts obtained by propagation of I_{obs} using T_{env} are shown as consequences to `User`: they do not need to be verified by observing the environment. This is safe because the propagated facts are true in the environment of `User`, by the definition of T_{env}.
- I^a_{obs}: The new environmental facts obtained by propagation of the state using T_{all} are shown as prerequisites of his decisions, to be verified by observing the environment. Indeed, there is no guarantee that they are true in the environment faced by `User`. The observation of the environment may occur at a later stage of the interactive process. If the observation does match the value obtained by propagation, `User` adds it to the state I_{obs}; the state will remain consistent. If the observation does not match, adding it would make the state inconsistent: `User` must first retract a decision to allow the entry, as explained in Section 3.3.2 on page 41.
- I^a_{dec}: The new decision facts obtained by propagation of the state using T_{all} are shown as consequences to `User`. `User` cannot just change such a value, since that would cause inconsistency. If `User` wants to change the propagated value, they first need to retract decisions that led to the propagation.

Example 17. *Returning to Example 15 on page 40, the optimal propagation of the state* ({*low_rent=**false***, *social_habitat=**false***}, {}) *is* $I^e_{env} = \{\}, I^a_{env} = \{\}$ *and* $I^a_{dec} = \{$*Registration_Type=Other*, *tax_rate*=10$\}$*: no observation needs to be verified. On the other hand, the optimal propagation of* $(\{\}, \{tax_rate = 7\})$ *is* $I^e_{env} = \{\}, I^a_{env} = \{$*low_rent=**true***$\}$ *and* $I^a_{dec} = \{$*Registration_Type=Modest*$\}$ *where* {*low_rent=**true***} *needs to be verified.*

3.3.4 Partial solutions

We now consider other forms of acceptable solutions. For example, it may not be necessary to observe the environment fully. In addition, some decisions may be unnecessary, in the sense that they can be made arbitrarily without impacting the result. Finally, the user may postpone some decisions when they have the guarantee that a solution exists.

We define definite and contingent solutions. A *definite* solution entails the satisfaction of T_{sol}, assuming T_{env}:

Definition 8. *A* definite solution *of the split model expansion problem $\mathcal{SMX}$ is a state (I_{obs}, I_{dec}) such that (i) I_{env} is an expansion of I_{obs} (i.e., $I_{env} \in \widehat{I_{obs}}$) and (ii) T_{sol} is satisfied by every expansion I_o of I_{obs} that satisfies T_{env} and by every expansion I_d of I_{dec},*

$$\forall I_o \in \widehat{I_{obs}} : [\![T_{env}]\!]^{I_o} = \textbf{\textit{true}} \implies \forall I_d \in \widehat{I_{dec}} : [\![T_{sol}]\!]^{I_o + I_d} = \textbf{\textit{true}}$$

Notice that Definition 8 treats T_{env} and T_{sol} very differently. This is because even though we do not know the full structure I_{env}, we do know that it exists, that it is more precise than I_{obs}, and that it satisfies T_{env}. For this reason, we only need to consider expansions of I_{obs} that satisfy T_{env}. For such expansions, it should be the case that no matter how we make our decisions, we obtain a solution to the problem at hand.

Example 18. *A definite solution for the simplified real estate problem without any prior observation is* ({}, {*Registration_Type*=*Other*, *tax_rate*=10}). *Indeed, any sale can be registered with type Other and a tax rate of 10.*

Any consistent state can be expanded into a definite solution. Indeed, by definition, every consistent state can be expanded into a total solution, which is a definite solution. In practice, definite solutions are often partial, not total, structures. As soon as `User` has found a definite solution, they can stop making observations and decisions. Indeed, such a solution can always be expanded to construct total solutions of the problem, by further observing the environmental and making any decision for the remaining decision symbols.

In some cases, `User` only needs the guarantee that a solution is reachable in their environment, by making the appropriate remaining decisions in a later stage. Since they do not know their environment fully, establishing this guarantee requires considering every environment more precise than what is known:

Definition 9. *A* contingent solution *of the split model expansion problem $\mathcal{SMX}$ is a state (I_{obs}, I_{dec}) such that (i) I_{env} is an expansion of I_{obs} and (ii) T_{sol} is satisfied by at least one expansion I_d of I_{dec} for every expansion I_o of I_{obs} that satisfies T_{env}, i.e.,*

$$\forall I_o \in \widehat{I_{obs}} : ([\![T_{env}]\!]^{I_o} = \textbf{\textit{true}}) \Rightarrow \exists I_d \in \widehat{I_{dec}} : [\![T_{sol}]\!]^{I_d + I_o} = \textbf{\textit{true}}$$

Note that, if a problem class is universally solvable, the empty state is a contingent solution.

Example 19. *In the Real Estate Sale Tax problem, the empty state is a contingent solution. Indeed, the problem class is universally solvable: for every possible environment, there is a possible registration.*

Definition 10. *An* incomplete solution *of the split model expansion problem $\mathcal{SMX}$ is a state (I_{obs}, I_{dec}) such that (i) I_{env} is an expansion of I_{obs} and (ii) there is at least one expansion I_d of I_{dec} such that T_{sol} is satisfied in every expansion I_o of I_{obs} that satisfies T_{env}, i.e.,*

$$\exists I_d \in \widehat{I_{dec}} : \forall I_o \in \widehat{I_{obs}} : (\llbracket T_{env} \rrbracket^{I_o} = \textbf{\textit{true}}) \Rightarrow \llbracket T_{sol} \rrbracket^{I_d + I_o} = \textbf{\textit{true}}$$

An incomplete solution is also a contingent solution.

Example 20. *An incomplete solution for the simplified real estate problem without any prior observation is* $(\{\}, \{Registration_Type = Other\})$. *It is also a contingent solution. The state can be extended into a definite solution by adding the tax rate:*

$$(\{\}, \{Registration_Type{=}Other, tax_rate{=}10\})$$

Complexity The complexity of deciding whether a state is a contingent or definite solution is much higher than checking whether it is consistent and total. In the context of propositional logic, checking for the existence of definite solution is Π_1^p-complete (coNP-complete) while checking for contingent solution is Π_2^p-complete. In Section "Implementation", we propose a cheap approximate method to make this decision.

3.3.5 Relevance

We now turn our attention to the question of determining which observations are relevant for the search. We have shown that uninterpreted symbols in a definite solution (I_{obs}, I_{dec}) are *irrelevant* (or "do-not-care") in the following sense: I_{dec} extended with *any* interpretation for these symbols will satisfy T_{sol} in *any* possible environment that expands I_{obs} while satisfying T_{env}. In this section, we extend the notion of relevance to any state.

Definition 11. *A* $\leqslant_p$-minimal definite solution *is a definite solution that is not more precise than another definite solution. A* $\leqslant_p$-minimal contingent solution *is a contingent solution that is not more precise than another contingent solution. We denote the set of minimal definite (resp. contingent) solutions more precise than a state by* $MinDef(I_{obs}, I_{dec})$ *(resp.* $MinCont(I_{obs}, I_{dec})$*).*

Definition 12. *The symbol σ in Σ_{all} is* relevant *in the consistent state (I_{obs}, I_{dec}) if, and only if, it is interpreted in at least one $\leqslant_p$-minimal definite solution more precise than the state, i.e.,*

$$\exists I \in MinDef(I_{obs}, I_{dec}) : \sigma \in \Sigma_I$$

where Σ_I is the set of symbols interpreted in I.

By our definition, all symbols interpreted in the state are relevant. As soon as `User` enters a value for a relevant symbol, the state is updated and the list of relevant symbols can be re-computed.

Proposition 1. *If there are no uninterpreted relevant symbols in a consistent state, the state is a definite solution.*

Proof. Suppose (I_{obs}, I_{dec}) is a consistent state without uninterpreted relevant symbols. By the definition of consistency, there is a total solution (and hence, a definite) solution that is more precise than (I_{obs}, I_{dec}), meaning that the set of definite solutions that expand (I_{obs}, I_{dec}) is non-empty. As a consequence (using the fact that $\leqslant_p$ is a well-founded order on the set of states of a fixed and finite vocabulary), this set has at least one $\leqslant_p$-minimal element. Let (I'_{obs}, I'_{dec}) be such a $\leqslant_p$-minimal definite solution that expands (I_{obs}, I_{dec}). By Def. 12 on the previous page, all symbols on which (I'_{obs}, I'_{dec}) and (I_{obs}, I_{dec}) differ are relevant. Since, by our assumption, all relevant symbols are interpreted in (I_{obs}, I_{dec}) (and thus also in (I'_{obs}, I'_{dec})), the state and the definite solution are actually equal. □

Proposition 2. *If there are no uninterpreted relevant environmental symbols in a consistent state, the state is an incomplete solution.*

Proof. Similarly to the previous proof, suppose (I_{obs}, I_{dec}) is a consistent state without uninterpreted relevant environmental symbols. Let (I'_{obs}, I'_{dec}) be a $\leqslant_p$-minimal incomplete solution that expands (I_{obs}, I_{dec}). By the definition of relevance, all symbols on which (I'_{obs}, I'_{dec}) and (I_{obs}, I_{dec}) differ are relevant. Since, by our assumption, all relevant environmental symbols are interpreted in (I_{obs}, I_{dec}) (and thus also in (I'_{obs}, I'_{dec})), I_{obs} and I'_{obs} are equal. Since the state differs from the incomplete solution only by the decisions and is consistent, it is an incomplete solution too. □

Such a state can be completed into a definite solution without making any additional observation.

These properties allow `System` to alert `User` when a definite (resp. incomplete) solution has been found. In propositional logic, the problem of computing whether a symbol σ is relevant in some state is Σ_3^p-complete. While computing the set of relevant symbols is computationally expensive, it can be approximated cheaply, as discussed in the next section.

Example 21. *When* {*low_rent*=***false***}*, licensed_seller is irrelevant. Indeed, Registration_Type cannot be Social nor Modest, and the only* $\leqslant_p$*-minimal definite solution is* {*low_rent*=***false***, *tax_rate*=10, *Registration_Type*=*Other*}*, in which licensed_seller is uninterpreted.*

3.4 Implementation: the Interactive Consultant

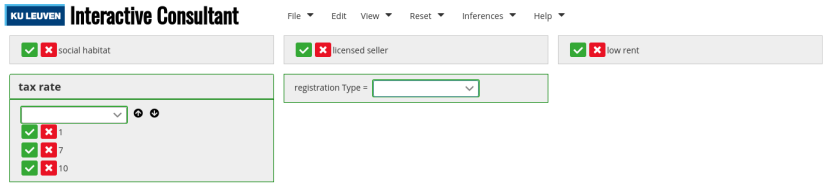


Figure 3.1: Screenshot of the Interactive Consultant

We have developed a decision support tool that embodies our proposal, called the Interactive Consultant. Given two vocabularies $(\Sigma_{env}, \Sigma_{sol})$ and two theories (T_{env} and T_{sol}), the tool automatically generates a user interface that allows the user to follow the proposed interactive process, with the benefit of computer-generated guidance. A screenshot is shown in Figure 3.1. The tool and its source code are available online.[2]

The user enters an observation or a decision by clicking on buttons or entering values in entry fields. He can do so in any order. The interactive advisor 1) updates the state, 2) propagates the state using T_{env}, 3) propagates again using $T_{env} \wedge T_{sol}$, and 4) determines the relevant symbols. The user interface is then updated to reflect the attributes and state of symbols using the graphical cues listed in Table 3.1 on the next page.

When there are no *environmental* symbols that are both relevant and uninterpreted, the solution is known to be a contingent, incomplete solution for the environment, by Proposition 2 on the facing page. When there is no symbol that is both relevant and uninterpreted, the solution is known to be a definite solution for the environment, by Proposition 1 on the preceding page, and the user can stop observing and making decisions.

The tool accepts theories written in FO[·], and it uses the IDP-Z3 reasoning engine. The reasoning engine has been extended with the reasoning capabilities required for interactively solving split model expansion problems. The new inferences are further described below.

[2] `https://interactive-consultant.idp-z3.be/` and `https://gitlab.com/krr/IDP-Z3`

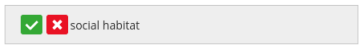
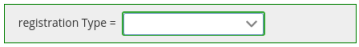
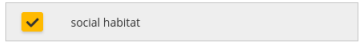
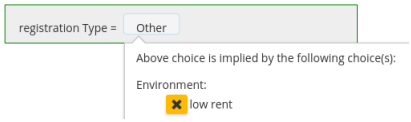
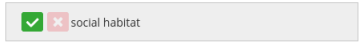
	A relevant environmental symbol is in a box with a grey border line
	A relevant decision symbol is in a box with a green border line
	An entry is shown in orange
	An irrelevant symbol is in a box with a white background
	A decisional consequence can only be edited after retracting a prior entry
	An environmental consequence that must be verified by observing the environment

Table 3.1: Graphical cues in the Interactive Consultant

Propagate Propagation is often performed by repeated satisfiability testing (Janota et al., 2015), i.e., by checking every ground atom to see if it is a consequence of the theory and user input. For performance reason, these checks can be run in parallel (Schneeweiss & Hofstedt, 2011). Instead, IDP-Z3 uses only one call to the `consequences` API of Z3 (Bjørner et al., 2018) to determine the ground atoms that are consequences of the state. This API uses conflict analysis to determine these consequences (Marques-Silva & Sakallah, 1999), and is faster than iterative satisfiability testing per our experiments.

The response time of the system after the user asserts or retracts a fact mostly depends on the speed of the propagation reasoning task. We improve the speed of propagation in the IC by using incremental propagation, i.e., by reducing the number of ground atoms to be checked, as follows:

- When new facts (observations or decisions) are asserted by the user, previously propagated atoms do not need to be checked again: indeed, their propagated value will not change.
- When facts are retracted by the user, atoms that were not consequences

of the theory and user input will remain so, and do not need to be checked again.

When the user opens a drop-down, IDP-Z3 populates the drop-down with the possible values of the symbol. These values are determined by using the propagation algorithm limited to this symbol (again for performance reasons).

Explain a consequence Given a theory T, a state S and a literal L obtained by propagation, IDP-Z3 computes an explanation for L in the form of a minimal set of sentences in $T \cup S \cup \{\neg L\}$ that is inconsistent. To perform this inference, IDP-Z3 uses the "unsat_core" API of Z3.

The translation of definitions using an equivalence, as described in Section 2.3 on page 24, is not ideal for the explanation inference: its result will refer to the equivalence as a whole, and not to the individual rule that applies. To address this issue, an alternative translation of the definition is used for the relevance inference: each rule is translated into one sentence, and a completion sentence is added, as illustrated for the `connected` relation (introduced in Listing 2.4 on page 27).

```
∀ c1, c2: connected(c1,c2) ⇐ railtrack(c1,c2).
∀ c1, c2: connected(c1,c2) ⇐ railtrack(c2,c1).
∀ c1, c2: connected(c1,c2) ⇒ railtrack(c1,c1)
                           ∨ railtrack(c2,c1).
```

(This alternative translation cannot be used in the other inferences without adversely impacting response time.)

Determine relevant symbols Because relevance, as defined in Section 3.3.5 on page 47, is expensive to compute, we approximate it using a 2-step process: 1) we compute propagation in state (I_{obs}, I_{dec}) as discussed in Section 3.3.3 on page 44 and create the theory C consisting of the propagated literals;[3] 2) we replace atoms occurring in the grounded[4] version of T_{all} by their value in the state (including propagated values), if any, and we simplify the formula using the laws of logic and arithmetic, to obtain T^*_{all}. Then, $C \cup T^*_{all}$ and T_{all} are equivalent in that state. The symbols that do not occur in theory $C \cup T^*_{all}$ cannot be relevant: indeed, changing their value cannot change the truth-value of T_{all} in the state. Thus, theory $C \cup T^*_{all}$ contains all the symbols that are

[3] I.e., if the T_{all}-propagation contains $\{p = \mathbf{true}, q = \mathbf{false}\}$, theory C contains $\{p.\neg q.\}$.

[4] The grounded version is obtained by expanding quantifications and aggregates over their (finite) domain.

relevant in the state; it may also contain some irrelevant symbols, depending on how thoroughly the propagation and simplification is performed, but this is acceptable in practice. Since Step 1 is computed for the reasons given in Section 3.3.3 on page 44, the additional cost of computing relevance is only determined by Step 2, and can be very low and still be useful to the user.

Definitions stated in the theory require special care when determining relevance. A defined atom can be made irrelevant by replacing it by its definiens in the theory: the transformed theory will be equivalent (except for the defined symbol), and the defined atom will not occur in it anymore. Yet, defined atoms are often relevant for the user as it abstracts complexity in a useful way. Thus, we distinguish definitions from the other logical sentences in the theory. We consider a ground atom relevant if and only if (i) it is important for the user to know its value, or (ii) it occurs in a simplified logical sentence, or (iii) it occurs in the simplified definition of a ground atom previously considered relevant. When a ground atom satisfies (iii), it contributes to the justification of another ground atom considered relevant, or of its negation. Hence, this method has many similarities with the justification-based method proposed by Jansen et al. (2016) to determine relevance.

3.5 Experimental evaluation

To validate our approach, we have configured the Interactive Consultant for the motivating example 2 (Section 1.1.2 on page 2). The source code for the Interactive Consultant is available on GitLab.[5] The tool is also available online to facilitate reproduction.[6] Our informal tests demonstrated that the system effectively assists the user by correctly determining which hypotheses must be verified, and which observations are relevant.

We then performed experiments on the real estate legislation adapted from Deryck et al. (2019) (available online[7] to facilitate reproduction). The theory to represent the legislation has 27 environmental symbols and two decision symbols.

We created 5 problem instances by randomly choosing Σ_{env} structures that satisfy T_{env}. We simulated a real user by entering values in the user interface in a random order. We measured how many decisions and observations we had to make, using the two approaches:

[5] `https://gitlab.com/krr/IDP-Z3`
[6] `https://tinyurl.com/simpleRegistration`
[7] `https://tinyurl.com/BelgianRegistration`

- To simulate the traditional approach, we enter the environmental data in a random order, and at each step, propagation by T_{env} is performed. If the value of an environmental symbol is derived by this T_{env}-propagation, we do not have to enter it. After all environmental data is entered, we choose decision values randomly among those that remain possible, until we obtain a total solution.
- To simulate the approach we propose, at each step, we fill in one uninterpreted symbol, chosen randomly among the relevant ones, until we obtain a definite solution. If the symbol is environmental, we enter the value given by the problem instance; if it is a decision symbol, we choose a possible value at random. When the state becomes inconsistent, we retract a conflicting decision at random.

Table 3.2 shows that the user workload is significantly reduced when the system determines which observations are relevant, and which observations must be made in the environment to ensure the solution is definite, as we propose.

In the traditional approach, the user has to make an average of 26.4 entries to obtain a model;[8] with the approach we propose, the user only needs to make 11.6 entries to obtain a definite solution, a 56% reduction.

	Count of Observations and Decisions	
	Traditional	Our proposal
Sale 1	26	10
Sale 2	26	14
Sale 3	26	10
Sale 4	27	10
Sale 5	27	14
Average	26.4	11.6
Gain		-14.8 (- 56 %)

Table 3.2: The number of observations and decisions is greatly reduced with our proposal.

The response time on each entry is less than 0.1 sec. The use of the Interactive Consultant in real-world applications is further discussed below, in Section 3.7 on page 61.

[8]Depending on the order of data entry, the value of some environmental symbols may be derived by propagation of T_{env}: in that case, the user does not have to enter a value for them.

3.6 A generalization: Collaborative and Temporal Model Expansion

In this section, we discuss a generalization of the interactive approach presented earlier to accommodate collaborative decision-making at various time points in a problem domain.

Indeed, several people may collaborate in searching a solution to a model expansion problem. For example, a planner and a manager may collaborate on a nurse roster problem: the planner may exceptionally plan overtime hours for some nurses, but only upon acceptance by the manager.

Furthermore, in a split model expansion problem, the split of vocabularies and of theories into the environmental and decision kinds is fixed by the knowledge engineer when the knowledge base is created. This is inappropriate when the interactive assistant can be used at various points in time, as the following examples illustrates.

Consider the problem of maintaining a monthly nurse roster: before the beginning of the month, the allocation of nurses to daily shifts is not fixed yet, but, in the middle of the month, the allocations of past days cannot be changed anymore; instead, they can be verified by observation. In some sense, due to the progression of time, some originally decision variables/facts have changed into environmental variables/facts. Similarly, in the Real Estate Sale Tax example above, we have assumed that the decisions to be made concern the registration of the sale to the authorities: at that time, whether the seller is a licensed seller can be verified by observation. But the legislation can be used for other decisions at other time points, e.g., it can be used before the house is bought to help choose the house to buy, taking the tax burden into account. Before the choice of the house to buy is made, the status of the seller as a licensed seller cannot be verified by observation: instead, it is under the control of the decision maker in the sense that they may decide to buy houses only from licensed sellers. So, "the seller is licensed" can be seen as an environmental or decision variable, depending on the time of the decision.

The generalization is based on the following general principles:

- one can only observe facts in the past; all facts in the past are observable;
- each symbol is owned by (at least) one agent: the owner of a symbol cannot change facts in the past, but they can choose facts in the future;
- "Nature" is such an agent;

- agents (including "Nature") collaborate in the sense that they choose future values of symbols that satisfy a common theory.

The Knowledge Representation methodology to solve such problems again relies on abstracting the states of affairs into logic structures over a suitable vocabulary. The structures abstract not only space, but also time: for example, the Real Estate Sales Tax theory above applies to the sales of properties at various places, but also to successive sales of the same property. To make computations possible, knowledge engineers often abstract continuous time using discrete time points. The informal meaning of these abstract, discrete time points are real time points in the problem domain.

To allow the representation of temporal knowledge, we use a temporal vocabulary. A *temporal vocabulary* consists of:

- a set of types, including a *Time* type having a total order ($<$);
- a set of *static* predicate and function symbols (without any occurrence of type *Time* in their type signature);
- a set of *dynamic* predicate and function symbols, with exactly one occurrence of type *Time* in their type signature.

Example 22. *Here is a possible temporal vocabulary for a monthly nurse rostering problem:*

Listing 3.2: Temporal vocabulary for a monthly nurse rostering problem

```
vocabulary Nurse {
    type Time ≜ {1..30}
    type Nurse
    onDuty: Time × Nurse → 𝔹
    nrOfShifts: () → Int

    type Patient
    admission: Patient → Time
}
```

In this vocabulary, `nrOfShifts` *is a static symbol;* `onDuty` *and* `admission` *are dynamic symbols.*

A Σ-structure I consists of a domain and an interpretation of the symbols in vocabulary Σ. The *past (resp. future) structure* of I at time t_0 (a value in $Time^I$) is a partial structure obtained by restricting the interpretation of dynamic symbols in I to the time period strictly before t_0 (resp. on or after

t_0), while keeping the interpretation of types and static symbols unchanged. They are denoted by $I_{<t_0}$ and $I_{\geqslant t_0}$. The past and future structures of I change when t_0 changes. They contain the same information as I: one can re-construct I, given $I_{<t_0}$ and $I_{\geqslant t_0}$. Note that these structures are partial, as the following example shows.

Example 23. *Consider the following structure I over the vocabulary presented in previous Example 22.*

```
structure I:Nurse {
    Nurse ≜ {}.
    onDuty ≜ {}.
    nrOfShifts ≜ 4.

    Patient ≜ {Bob, Alice}
    admission ≜ {Bob → 1, Alice → 10}
}
```

This structure is total. In the structure $I_{<4}$, the interpretation of `admission` *is the partial function* `Bob → 1`*: it is undefined on* `Alice`*. Hence, the fact that the past interpretation of* `admission(Alice)` *at time 4 is undefined means that it occurs at a time after 4. We can see in this structure $I_{<4}$ that no patients were admitted at time 2 or 3. In some sense, this is a local form of the Closed World Assumption, for the knowledge of the past.*

At the time t_0 where a decision has to be made, we consider that the facts describing the environment are precisely the facts of the past in the actual state of affairs. We assume that all facts of the past are known to us or can be observed. The decision makers, including Nature, are free to choose the future facts, subject to the theory to be satisfied. This leads us to the following definition.

Definition 13. *The* temporal model expansion problem $\mathcal{TMX}$ *is a problem that takes as input a tuple $(\Sigma, T, D, Time, t_0, I_{<t_0})$ where T is a theory over temporal vocabulary Σ, t_0 is a value in Time (a subset of domain D), and $I_{<t_0}$ is a past structure over Σ and D at time t_0.*

A total solution *for such a problem is a total structure I over Σ and D whose past structure for t_0 is $I_{<t_0}$ and such that*

$$[\![T]\!]^I = \textbf{\textit{true}}.$$

A temporal model *optimization* problem is defined in a similar way.

Although $I_{<t_0}$ is a partial structure over Σ and D, it is the past structure of a solution, and thus, it contains all the information strictly before t_0. Hence, the

absence of a past fact in it entails that the past fact is false. As time progresses, i.e., as t_0 increases, new knowledge can be inferred from structure $I_{<t_0}$ even when the structure does not change, as the following example illustrates.

Example 24. *Consider a patient monitoring system that must raise an alarm when the patient has not been visited by a nurse in the last 30 minutes. The decision has to be made every minute. The problem domain can be formalized as follows:*

```
vocabulary monitoring {
    type  Time isa Int    // minutes
    visit: Time → 𝔹    // a nurse visit occurred at time t
    alarm : Time → 𝔹   // the alarm sounds at time t
}
theory {
    { ∀t: alarm(t) ←
        ∀ t1 in Time: t-30 ≤ t1 < t ⇒ ¬visit(t1).
    }
}
```

Structure $I_{<t_0}$ describes the environment at the time t_0 of a decision. The structure is updated every minute. At time 10, a decision for `alarm(10)` *has to be made given that $I_{<10}$ is* `visit≜{0}. alarm≜{}.`*, meaning that the only visit in the past was at time 0, and that more visits may happen, but only on or after time 10: the decision is* ***false****, meaning that the alarm must not sound. At time 31, a decision has to be made for* `alarm(31)` *given that $I_{<31}$ is* `visit≜{0}. alarm≜{}.`*: the decision is* ***true****, meaning that the alarm must sound (there were no visits in the last 30 minutes).*

A temporal problem is not always solvable, as the next example shows.

Example 25. *Consider the theory that says: (i) if the sun shines at time 0, I don't take my umbrella at time 1, (ii), if it rains at time 0, I take my umbrella at time 1. If the sun shines at time 0 and if it also rains at time 0, there is no solution.*

As explained in Section 3.3.2 on page 41, when a problem has many solutions in a given environment, and when the decision makers may prefer some solutions over others for reasons that are not explicitly represented in theory T, then an interactive system can help them explore the set of solutions. Such systems are also useful when the environment, while fixed, is not fully known by the user at the start of the search, but can be observed as needed: in many cases, full observation of the environment is not required, as explained in Section 3.3.4 on page 45. In those cases, the interactive system can help the user make relevant observations.

To represent a state of the interactive system, we define:

Definition 14. *A state of the collaborative process is a tuple* $(S_{<t_0}, S_{\geqslant t_0})$ *such that* $S_{<t_0}$ *(resp.* $S_{\geqslant t_0}$*) is a past (resp. future) structure over* Σ*.*

$S_{<t_0}$ describes the observations of past facts, and $S_{\geqslant t_0}$ describes the tentative decisions made by the decision makers.

Definition 15. *A state* $(S_{<t_0}, S_{\geqslant t_0})$ *is* consistent *iff there is an expansion of the state that satisfies* T*:*

$$\exists I \in \widetilde{S_{<t_0} + S_{\geqslant t_0}} : [\![T]\!]^I = \textbf{\textit{true}}$$

Definition 16. *A* collaborative process *for a* $\mathcal{TMX}$ *problem with input* $(\Sigma, T, D, Time, t_0, I_{<t_0})$ *is a sequence* $(S^i_{<t_0}, S^i_{\geqslant t_0})_{0<i\leqslant n}$ *of* n *consistent states, such that, at each step, either the set of observations or the set of decisions is made either more precise or less precise, i.e., for any* $0 < i < n$*:*

- $(S^i_{<t_0} <_p S^{i+1}_{<t_0} \vee S^{i+1}_{<t_0} <_p S^i_{<t_0}) \wedge S^i_{\geqslant t_0} = S^{i+1}_{\geqslant t_0}$
- *or,* $S^i_{<t_0} = S^{i+1}_{<t_0} \wedge (S^i_{\geqslant t_0} <_p S^{i+1}_{\geqslant t_0} \vee S^{i+1}_{\geqslant t_0} <_p S^i_{\geqslant t_0})$.

In a collaborative process, anyone can make the state less precise, and anyone can make $S_{<t_0}$ more precise following an observation, but a new fact can be entered in $S_{\geqslant t_0}$ only by its owner (unless it is "Nature", in which case, anyone can enter it). Note that, if observations are accurate, there is no need to retract observations.

Similarly to the interactive process for an $\mathcal{SMX}$ problem, `System` helps `User` update the state while keeping it consistent. For example, if an agent needs to make $S_{<t_0}$ more precise following an observation, and if this change would make the state inconsistent, `System` asks `User` to retract a decision first, so that the entry of the observation does not make the state inconsistent anymore.

`System` also propagates the state, i.e., derives new facts from the facts in the state and from theory T. In general, derived facts about the past need to be verified by observation, but this is not always the case. We first consider the case where observations are accurate and the problem is known to be solvable.

(i) If a fact in the past is derived only from $S_{<t_0}$, it does not have to be observed: indeed, if the observation conflicted with the derived value, either an observation would be incorrect or the problem would not be solvable, contradicting our assumption;

(ii) If a fact in the past is derived from $S_{\geqslant t_0}$ (possibly together with $S_{<t_0}$), then it has to be verified by observation; indeed, there is no guarantee that it is a fact in $I_{<t_0}$; if the observation conflicts with the propagation, some future facts must be retracted before the observation can be entered;

(iii) When a fact in the future is derived, it cannot be observed; instead, in a collaborative setting, its owner must be asked for approval. If the owner is "Nature", no approval is needed.

If observations are accurate and the problem is not known to be solvable, then item (i) must be refined as follows:

(ia) If a fact a occurring at time $t_a < t_0$ is derived only from facts occurring strictly prior to t_a, the fact a does not have to be observed. Indeed, the owner of fact a was free to choose the derived value at time t_a, and he must have chosen that value to satisfy T.

(ib) If a fact a occurring at time $t_a < t_0$ is derived only from facts prior to t_0, and some of those facts occurred on or after t_a, the fact a has to be verified by observation; if the observation conflicts with the derived value, the problem is unsolvable.

Example 26. *Continuing Example 25, assumes that, at time 2, I observe that it rained at time 0. Then, at time 1, I must have freely chosen to take my umbrella at time 1 to satisfy theory T, and that past fact does not have to be observed. However, the theory and the fact that it rained at time 0 also entail that the sun did not shine at time 0, and that fact must be verified by observation. If it is observed that it shone at time 0, the problem is unsolvable.*

One can see the correspondence with the previous sections, shown in Table 3.3. The concept of partial solutions and of relevance, introduced in Sections 3.3.4 and 3.3.5, can be easily adapted to the temporal setting.

Interactive	Temporal
$\Sigma_{env}, \Sigma_{sol}$	Σ
T_{env}, T_{sol}	T
I_{env}, I_{obs}	$I_{<t_0}, I_{\geqslant t_0}$
I_{obs}, I_{dec}	$S_{<t_0}, S_{\geqslant t_0}$

Table 3.3: Correspondence between Interactive Model Expansion and Temporal Model Expansion

We now illustrate some aspects of temporal vocabularies, using this example:

Listing 3.3: Temporal vocabulary for the Real Estate Sales Tax problem

```
vocabulary Tax {
    type Time ≜ {time_of_building, time_of_licensing,
                 time_of_sale,time_of_registration}
    social_habitat  : {time_of_building} → 𝔹
    licensed_seller : {time_of_licensing} → 𝔹
    low_rent        : {time_of_sale} → 𝔹

    type Registration_Type ≜ {Social, Modest, Other}
    type Tax_Rate ≜ {1, 7, 10}

    tax_rate : {time_of_registration} → Tax_Rate
    registration_Type : {time_of_registration}
            → Registration_Type
}
```

The temporal aspect of the problem domain is discretized into four time points of interest. The order in which they are enumerated defines the total order of type `Time`. The predicates and functions have one `Time` argument; their domain is a singleton subset of the `Time` type. The informal meaning of the argument is that these facts are decided at those specific time points only. For example, the `tax_rate` is decided at the time of registration. If that time is in the past, the tax-rate is fixed and can be observed: if it is in the future, it can be chosen, subject to the theory.

Because `licensed_seller` has a truth-value only when its argument is `time_of_licensing`, and because a sentence must have a truth value, the badly formed `licensed_seller()` expression occurring in sentences can be accepted as a convenient way to unambiguously formulate the well-formed but longer `licensed_seller(time_of_licensing)` expression. For example, one can write:

```
licensed_seller() ∧ social_habitat() ⇐
    registration_Type() = Social.
```

This way, the theory in Listing 3.1 on page 40 can use the temporal vocabulary described in Listing 3.3.

A decision can be made at any of the four time points, given all the facts occurring before it. For example, at the time of building a property, one can choose to build a social habitat (or another type of property), and explore the future tax burden under various scenarios. At the time of registration, however, one can only choose the type of registration and the tax rate, given all the other facts.

Reasoning	IDP-Z3	Z3	clingo 5	IDP3
Model checking	✓	✓	✓	✓
Model expansion	✓	✓	✓	✓
Possible values	✓			✓
Propagation	✓	-*	-**	✓
Explanation (unsat)	✓	-*	-**	✓
Optimization	✓	✓	✓	✓
Relevance	✓		-**	

(*) Requires change to the knowledge representation
(**) Requires custom code

Table 3.4: Comparison of reasoning capabilities of solvers.

A difficulty arises when some courses of events follow parallel paths: for example, the path for building the house is parallel to the path for obtaining the seller license. In this case, it is not clear whether `time_of_building` should be before or after `time_of_licensing`. A possible solution is to allow two possible time points for licensing:

```
licensed_seller: ( { time_of_licensing_before_building,
                     time_of_licensing_after_building}
                 ) → 𝔹
```

This may lead to additional complications, however, and better representations may be needed.

3.7 Discussion

IDP-Z3 natively supports more reasoning tasks than other solvers, as shown in Table 3.4.

IDP-Z3 and the Interactive Consultant are now in use in higher education (in the exercise sessions of the Knowledge Representation course of KU Leuven), as well as in industry to solve real-life configuration problems.

Aerts et al. (2022) describe the creation of an IDP-based knowledge base system for the design of machine components, implemented in partnership with a multinational company. This company employs engineers worldwide to conceive "design-to-order" components. Before the collaboration, each engineer followed their own *ad hoc* design process, mostly based on their own experience and preferences. This approach has multiple downsides: (a) designing a component

consumes a large amount of time, (b) engineers may choose suboptimal designs, and (c) if a senior engineer leaves, a great deal of knowledge is lost by the company.

To overcome these issues, the company wanted to boost their engineers with a *Component Designer* tool. Originally, the AutoConfig system was used to build it. However, its limitations led them to later switch to the Interactive Consultant.

The design knowledge was formalized in FO[·] through a series of knowledge extraction workshops. Each workshop spanned a few days, during which both knowledge and design engineers were present. The resulting Knowledge Base contains 10 parameters describing 60 different materials (such as a maximum temperature of steel) and 27 parameters describing 31 components (such as torque and maximum pressure), and 42 sentences to describe acceptable configurations.

Overall, the company and its engineers are very positive about the tool. Besides a reported daily time-save of up to 30 minutes for each engineer, they report other benefits to its usage. First, it leads to more "first-time right" designs, which lowers time and cost. Second, for new engineers, the tool serves as a learning tool, allowing them to indirectly learn from the knowledge of the more experienced engineers. For more experienced engineers, on the other hand, the tool is used to challenge their assumptions: when in doubt, they can swiftly verify if their initial ideas are correct. Lastly, with their knowledge captured in a Knowledge Base, engineers leaving will not result in loss of knowledge for the company.

A similar engineering-focused use case is described in Jordens et al. (2022); Vandevelde et al. (2022). Here, a company that employs adhesive experts wanted a tool to support them in selecting correct adhesives for gluing operations. The KB was formalized during multiple knowledge workshops through collaboration between adhesive experts and knowledge engineers. The adhesive experts now interact with the knowledge base through the Interactive Consultant. They especially appreciate being able to explore the problem domain interactively: each time they enter a design requirement, they observe the number of suitable adhesives going down, showing them that they are converging towards a solution. Selecting adhesives is now faster, with a reduction in selection time from three hours to five minutes.

We now discuss the computational efficiency of IDP-Z3 in these case studies. Table 3.5 on the next page shows various metrics for each use case:

- symb.: the number of symbols in the vocabulary;

Use case	symb.	size	sent.	load	values	respond	explain
Real Estate Sales Tax	31	31	15	0.3	0.1	0.1	0.1
Component designer	92	227286	42	14	6.5	1.5	11.6
Adhesive selection	136	2061	154	5.3	0.17	1.2	0.2

Table 3.5: Various performance metrics of IDP-Z3. All metrics to the right of the vertical bar are in seconds.

- size: the size of a model, i.e., the sum of the cardinality of the domain of each predicate and function symbol (they all have a finite domain);
- sent.: the number of sentences and definitional rules in the theory;
- load: the number of seconds needed to load the knowledge base in the IC;
- values: the typical number of seconds to obtain possible values in a list box;
- respond: the typical number of seconds needed to respond to a new data entry;
- explain: the typical number of seconds needed to explain a consequence.

Performance is measured on an Intel® Core™ i7-8850H CPU @ 2.60GHz × 12, 16 GB of RAM, on Ubuntu 22.04.02, CPython 3.11.5 and IDP-Z3 0.10.12.

The table shows that the load time can exceed 10 seconds for KB with large models. This is not ideal, but still acceptable in the use case presented. The delay is due to the transformation of the FO(·) KB into the equivalent grounded FO formula that is submitted to Z3: the transformation is performed at initial load, by code written in Python, a language not known for its speed. Load time could be improved by rewriting the transformation in a faster language, or by loading the pre-computed result of the transformation directly from storage.

The response time of the Interactive Consultant, on the other hand, is below 3 seconds, an adequate delay for interactive applications. This is achieved by retaining the internal state of IDP-Z3 between interactions. Our experience indicates that such retention is important in interactive applications based on IDP-Z3.

3.8 Personal contribution

This chapter is based on Carbonnelle et al. (2023d). The paper is the synthesis of many discussions with Profs. Denecker, Vennekens and Bogaerts. I conceived the approximate method to determine relevant symbols for FO($\cdot$) theories using standard solvers. I developed the Interactive Consultant, including all the inferences it uses. I conceived the experimental setup for its evaluation. Simon Vandevelde and other colleagues used the Interactive Consultant for the applications reported in the discussion section (Section 3.7 on page 61). I added the distinction between contingent and incomplete solutions, and I extended the split model expansion concept to obtain the collaborative temporal model expansion concept (Section 3.6 on page 54), with input from Prof. Denecker.

Chapter 4

Lifted Model Expansion

4.1 Introduction

In Section 3.2 on page 37, we presented configuration problems as an important case of model expansion. Methods to solve configuration problems do not scale well when the number of components increases, and various heuristics and machine learning techniques have been used to improve performance (Schenner & Taupe, 2017; Guo et al., 2023). Configurations often have components that are interchangeable. They are the source of many redundancies in the search space that negatively impact performance.

We observed that solutions to configuration problems can be compressed in what we call a "lifted model", and that the solution of the original problem can be obtained by expanding the lifted model. This suggested that configuration problems, and, more generally, model expansion problems, could be solved in the compressed domain: if the compressed domain is significantly smaller, it could lead to better performance.

As a toy example, consider the following pigeonhole problem: "Given 10 pigeons and 5 pigeonholes, assign each pigeon to a pigeonhole such that each hole has exactly 2 pigeons." With the appropriate vocabulary, a solution for this problem

is (Figure 4.1):

$$Hole = \{ h_1, \ldots, h_5\} \tag{4.1}$$

$$Pigeon = \{p_1, \ldots, p_{10}\} \tag{4.2}$$

$$isIn = \{ (p_1, h_1), (p_2, h_2), \ldots, (p_5, h_5), \tag{4.3}$$
$$(p_6, h_1), (p_7, h_2), \ldots, (p_{10}, h_5)\}$$

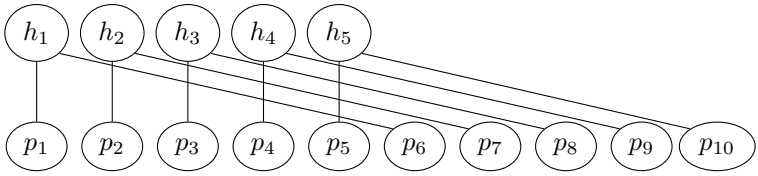


Figure 4.1: Concrete solution of the Pigeonhole problem

Note the symmetries in the solution: another solution is obtained by exchanging 2 pigeons or 2 holes in the interpretation of *isIn*.

This solution can be compressed to (Figure 4.2):

$$Hole = \{\mathbf{h}_1\} \tag{4.4}$$

$$Pigeon = \{\mathbf{p}_1\} \tag{4.5}$$

$$isIn = \{(\mathbf{p}_1, \mathbf{h}_1)\} \tag{4.6}$$

$$mul = \{\mathbf{h}_1 \mapsto 5, \mathbf{p}_1 \mapsto 10\} \tag{4.7}$$

where the *mul* function (for *mul*tiplicity) indicates how many concrete domain elements each "lifted" domain element represents, and thus allows the domain compression. The lifted hole represents 5 concrete holes, and the lifted pigeon represents 10 concrete pigeons.

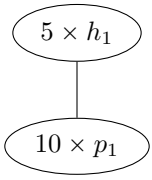


Figure 4.2: Lifted solution. The number in a node is the multiplicity

Even though the compressed structure only has 2 domain elements ($\mathbf{p}_1, \mathbf{h}_1$, excluding the natural numbers), it contains enough information to allow us to *expand* it into a model isomorphic to the original model, by evenly distributing the 10 pigeons in the 5 pigeonholes. Following the theory we develop in Section 4.3 on page 70, the expansion is as follows:

$$\mathbf{h}_1 \rightsquigarrow h_1, \ldots, h_5 \tag{4.8}$$

$$\mathbf{p}_1 \rightsquigarrow p_1, \ldots, p_{10} \tag{4.9}$$

$$(\mathbf{p}_1, \mathbf{h}_1) \rightsquigarrow \{(p_1, h_1), \ldots, (p_5, h_5), (p_6, h_1), \ldots, (p_{10}, h_5)\} \tag{4.10}$$

Furthermore, in Section 4.3.2 on page 77, we present a translation of sentences in typed first-order logic extended with aggregates into sentences that allow domain compression. We show that a sentence in that language, called FO(Type, Aggregate), is equisatisfiable with its translation: if either one is satisfiable, the other one is too. This analysis allows us to solve model expansion problems in the compressed domain. The novel method operates in two steps: (i) the sentence to be satisfied is automatically translated into a "lifted" sentence over the "lifted" vocabulary; (ii) satisfiability of the lifted sentence is checked by step-wise growing the compressed domain[1] until a satisfying structure is found. Crucially, this satisfying structure can always be expanded into a model of the original sentence with the given number of concrete elements in each type.

Notice that the lifted model for a variation of the pigeonhole problem above with 100 times more pigeons and holes has the same number of lifted domain elements as the base case: the multiplicity of each lifted domain element is simply multiplied by 100. As a result, and unlike traditional methods, our method solves that pigeonhole problem in constant time with respect to the domain size (excluding the natural numbers).

In the field of probabilistic inferences, an algorithm is called "domain lifted" if it runs in time polynomial in the size of the domain (de Salvo Braz et al., 2005). We borrow this term to name our method.

We evaluate our method by comparing the time needed to find solutions for generative configuration problems discussed in the literature. In *generative* configuration problems, the number of components in the solution is not known in advance. Our evaluation shows that our method has significantly better performance than the traditional one for problem formulations with symmetries and with a preponderance of unary predicate and function symbols.

We first recall some concepts about permutations.

[1] An iterative method is required because the size of the compressed domain is not known in advance.

Permutations and orbits A *permutation* of a set is a bijection from that set to itself. We denote a permutation by π, and its inverse by π^{-1}. The *identity* permutation, π^0, maps every element of the set to itself. The *order* of a permutation is the smallest positive number n such that π^n is the identity permutation. Hence, the n-th permutation of an element is equal to its "n-th modulo the order" permutation.

Cycles are permutations that map elements in a cyclic fashion, e.g., it maps element a to b, b to c, and c to a. A cycle is denoted by $(d^1 d^2 \cdots d^n)$, to express that $\pi(d^i) = d^{i+1}$ for $1 \leqslant i < n$, and $\pi(d^n) = d^0$. The orbit of the cycle is the set $\{d^1, d^2, \cdots, d^n\}$. The *length* of a cycle is the size of its orbit. A permutation over a finite set has a unique decomposition into *disjoint cycles*. A permutation is then denoted by its cycles, e.g., $(abc)(de)$. Its order is the least common multiple (*lcm*) of the length of its cycles. Cycles of length one are often omitted in the cycle notation; the elements they contain are fixed points.

The permutation of a tuple of elements $\bar{d}$ is denoted by $\pi(\bar{d})$ and is equal to $(\pi(d_1), ..., \pi(d_n))$. The π-*orbit* of an element d (resp. of a tuple of elements $\bar{d}$) is the set of its repeated permutations $o_\pi(d) = \{\pi^i(d) \mid i \in \mathbb{N}\}$ (resp. $o_\pi(\bar{d}) = \{\pi^i(\bar{d}) \mid i \in \mathbb{N}\}$). The size of the π-orbit of a tuple is the least common multiple of the size of the π-orbits of its elements.

4.2 State of the art: symmetry breaking

Symmetries are important in model expansion because they can be exploited to reduce the search space. The idea is to never consider a tentative solution that is symmetric to another tentative solution that has already been considered because we know that one is a model only when the other is a model. Realizing this idea is the topic of a large area of research.

The concept of symmetry can be defined in two different ways: as a property of the sentence to be satisfied, or as a property of the solution set (Gent et al., 2006). We choose to define symmetries as a property of the solution set, as done by Devriendt et al. (2016). We use Γ_D to refer to the set of structures having domain D.

Definition 17 (Symmetry)**.** *A* structure transformation *is a mapping from Γ_D to Γ_D. A structure transformation σ is a* symmetry *for Σ-sentence ϕ if and only if, for any Σ-structure I, I is a model of ϕ if and only if $\sigma(I)$ is a model of ϕ.*

A *domain permutation* π is a permutation of the domain of structure I that maps numbers to themselves, and other domain elements in T^I to domain

elements in T^I. Since numbers are mapped to themselves by the permutation, and since types besides $\mathbb{Q}$ are finite, the permutation is composed of cycles.

A domain permutation *induces a structure transformation.* The interpretation in the transformed structure, denoted $\pi(I)$, is defined as follows:

- the type of domain element d in $\pi(I)$ is its type in I;
- a tuple $\bar{d'}$ is in the transformed interpretation of predicate symbol p if and only if $\pi^{-1}(\bar{d'})$ is in the original interpretation of p:

$$p^{\pi(I)} = \{\bar{d'} \mid \pi^{-1}(\bar{d'}) \in p^I\} = \{\pi(\bar{d}) \mid \bar{d} \in p^I\} \tag{4.11}$$

- a tuple $(\bar{d'} \mapsto d')$ is in the transformed interpretation of function symbol f if and only if $\pi^{-1}(\bar{d'}) \mapsto \pi^{-1}(d')$ is in the original interpretation of f:

$$f^{\pi(I)} = \{(\bar{d'} \mapsto d') \mid (\pi^{-1}(\bar{d'}) \mapsto \pi^{-1}(d')) \in f^I\} \tag{4.12}$$

$$= \{(\pi(\bar{d}) \mapsto \pi(d)) \mid (\bar{d} \mapsto d) \in f^I\} \tag{4.13}$$

Proposition 3. *Any structure transformation induced by a domain permutation is a symmetry for any sentence.*

This can be proven by structural induction over the syntactic structure of the sentence. We say that $\pi(I)$ is isomorphic with I.

Definition 18 (Automorphism)**.** *A domain permutation π that transforms a structure into itself, i.e., such that $I = \pi(I)$, is an* automorphism *of the structure.*

In an automorphism, the following properties hold for each predicate symbol $p : \bar{T} \to \mathbb{B}$ and each function symbol $f : \bar{T} \to T$.

$$\forall x \in T : \forall i \in \mathbb{N} : \pi^i(x) \in T \tag{4.14}$$

$$\forall \overline{x} \in \overline{T} : \forall i \in \mathbb{N} : p(\overline{x}) \Leftrightarrow p(\pi^i(\overline{x})). \tag{4.15}$$

$$\forall \overline{x} \in \overline{T} : \forall i \in \mathbb{N} : \pi^i(f(\overline{x})) = f(\pi^i(\overline{x})). \tag{4.16}$$

Example 27. *In our pigeonhole problem (Figure 4.1), $(h_1 h_2)$ is a domain permutation that induces a structure transformation that is a symmetry for our sentence, but it is not an automorphism because $isIn^{\pi(I)}$ has tuple $\pi(p_1, h_1) = (p_1, h_2)$ which is not in $isIn^I$. On the other hand, permutation $(h_1 h_2)(p_1 p_2)(p_5 p_6)$ is an automorphism : for example, tuple $\pi(p_1, h_1) = (p_2, h_2)$ is in $isIn^I$.*

The automorphism π describes a symmetry of the structure I. Indeed, π describes a permutation of the domain of I that leaves the structure unchanged with respect to the satisfiability relation: I satisfies a sentence if and only if its permutation also satisfies the sentence.

Every structure has at least one automorphism: the identity domain permutation. Note that an automorphism maps the interpretation of any constant to itself, i.e., $c^I() = \pi(c^I())$. Thus, the length of the cycle containing $c^I()$ is 1.

There are three complementary approaches to exploiting symmetries:

- reformulate the problem to reduce symmetries;
- add symmetry breaking constraints before the search starts (Crawford et al., 1996);
- adapt the search procedure during the search, to take symmetries into account.

We refer to Gent et al. (2006) for an overview of these methods.

The first approach is more an art than a science: there is no general method available to apply it. The method we present below is an automatic reformulation method that has benefits in some applications, but that is not universally beneficial.

The following example illustrates the second method.

Example 28. *The problem is to find 3 distinct positive numbers, denoted a, b, c, whose sum is 6. Each solution assigns a permutation of the tuple $(1, 2, 3)$ to a, b, c. To reduce the solution set, one can impose that $a < b < c$: only one solution remains.*

4.3 Theoretical contribution

4.3.1 Lossless compression of structures

In this section, we discuss how, and when, a concrete structure with symmetries can be compressed into a lifted structure over a smaller domain without loss of information, and how a lifted structure can be expanded into a concrete one.

We introduce the new concept of *backbone*, which plays a critical role in the compression that we propose. Essentially, a *backbone* for an automorphism

of I is a set of domain elements obtained by picking one element in each cycle, such that the interpretation of any symbol can be reconstructed from its interpretation restricted to the backbone, by applying the automorphism repeatedly. Thus, the interpretation of the symbols restricted to the backbone is a *pattern*, and the permutations of that pattern form the full interpretation. Formally:

Definition 19 (Backbone)**.** *A* backbone *for an automorphism π of I is a subset S of the domain of I such that:*

- *each cycle C of π has exactly one element in S;*
- *for each predicate $p/n \in \Sigma$, p^I is the union of the π-orbits of the tuples in $p^I \cap S^n$, i.e.,*

$$p^I = \bigcup_{\bar{d} \in p^I \cap S^n} o_\pi(\bar{d}) = \{\pi^i(\bar{d}) \mid \bar{d} \in p^I \cap S^n, i \in \mathbb{N}\} \tag{4.17}$$

- *for each function $f/n \in \Sigma$, f^I is the union of the π-orbits of the tuples in $f^I \cap (S^n \times S)$, i.e,*

$$f^I = \bigcup_{(\bar{d} \mapsto d) \in f^I \cap (S^n \times S)} o_\pi(\bar{d} \mapsto d) \tag{4.18}$$

$$= \{\pi^i(\bar{d} \mapsto d) \mid (\bar{d} \mapsto d) \in f^I \cap (S^n \times S), i \in \mathbb{N}\}$$

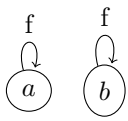
Example 29. *For the pigeonhole example in the introduction of the paper, a backbone of automorphism $(p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{10})(h_1h_2h_3h_4h_5)$ is $S = \{p_1, h_1\}$. Indeed, the images of (p_1, h_1) obtained by repeatedly applying the permutation are:*

$$\{(p_1, h_1), \ldots, (p_5, h_5), (p_6, h_1), \ldots, (p_{10}, h_5)\}$$

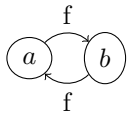
Another backbone is $S = \{p_2, h_2\}$.

In all structures, the set of domain elements is a *trivial* backbone for the identity automorphism. However, not all automorphisms have a backbone, as the following example shows.

Example 30. *Let $\{a, b\}$ be a (concrete) domain with one type T. In structure I_1 (resp. I_2), function symbol $f : T \to T$ is interpreted as $\{a \mapsto a, b \mapsto b\}$ (resp. $\{a \mapsto b, b \mapsto a\}$, Figure 4.3 on the next page). Permutation (ab) is an automorphism of both structures. The only two subsets S satisfying the first condition of the definition of backbone for (ab) are $\{a\}$ and $\{b\}$. Since f^{I_1} can be reconstructed from $f^{I_1} \cap (S \times S)$ for both subsets S, both subsets are backbones of I_1. However, none is a backbone of I_2 (because $f^{I_2} \cap (S \times S) = \varnothing$ for both candidate sets S).*



(a) (ab) has a backbone in I_1



(b) (ab) has no backbone in I_2

Figure 4.3: Automorphisms with and without a backbone

A backbone enables us to lift a structure into a structure with a smaller domain, as we now describe.

Definition 20 (Lifted vocabulary)**.** *For a vocabulary Σ, its lifted vocabulary Σ_l consists of the symbols of Σ and, for any type T of Σ, a symbol $mul_T : T \to \mathbb{N}$, called the* multiplicity *function for T.*

We will drop the subscript T in the function symbol mul_T when this is unambiguous.

Definition 21 (Lifted structure)**.** *Let I be a Σ-structure with an automorphism π having the backbone S. A lifted structure L for backbone S of structure I is a Σ_l-structure such that:*

- *its domain is S, called the lifted domain;*
- *for each type symbol T, $T^L = T^I \cap S$;*
- *for each predicate symbol p/n, $p^L = p^I \cap S^n$;*
- *for each function symbol f/n, $f^L = f^I \cap (S^n \times S)$;*
- *for each $l \in S$, $mul^L(l) = |o_\pi(l)|$, the size of the π-orbit of l in I.*

Example 31. *Continuing the pigeonhole problem, the lifted structure for backbone $S = \{p_1, h_1\}$ is described by Equations 4.4 to 4.7 on page 66.*

Proposition 4. *For the identity automorphism of a structure I, the set $S = dom(I)$ is a backbone and the lifted structure L for backbone S is I extended with $mul(s) = 1$ for each $s \in S$.*

Given the lifted structure L for backbone S of structure I and the automorphism π on the concrete domain I, one can reconstruct I, i.e., one can *expand* the lifted interpretations of the type, predicate and function symbols, essentially by

closing them under repeated application of π. Formally,

$$exp_\pi(T^L) = \bigcup_{l \in T^L} o_\pi(l) = \{\pi^i(l) \mid l \in T^L, i \in \mathbb{N}\} \tag{4.19}$$

$$exp_\pi(p^L) = \bigcup_{\bar{l} \in p^L} o_\pi(\bar{l}) = \{\pi^i(\bar{l}) \mid \bar{l} \in p^L, i \in \mathbb{N}\} \tag{4.20}$$

$$exp_\pi(f^L) = \bigcup_{(\bar{l} \mapsto l) \in f^L} o_\pi(\bar{l} \mapsto l) = \{\pi^i(\bar{l} \mapsto l) \mid (\bar{l} \mapsto l) \in f^L, i \in \mathbb{N}\}. \tag{4.21}$$

Example 32. *Continuing the pigeonhole problem, the expansion of the lifted structure for backbone $S = \{p_1, h_1\}$ (Equations 4.4 to 4.7 on page 66) and for automorphism $(p_1 \cdots p_{10})(h_1 \cdots h_5)$ is the structure described by Equations 4.1 to 4.3 on page 66.*

In our approach, we need to find lifted structures that can be expanded into concrete ones. We observe that there is a simple construction to expand *any* lifted Σ_l-structure L with lifted domain S that results sometimes (but not always) in a concrete structure I with an automorphism π having backbone S. This construction begins with the expansion of the lifted domain. Any lifted domain element $l \in S$ is expanded to concrete domain elements $d^1, \ldots, d^n$ where l is identified with d^1 and n is equal to $mul^L(l)$. If the multiplicity of l is zero,[2] its expansion is the empty set. The expansion of any two distinct lifted elements must be disjoint. Formally:

Definition 22 (Expansion of lifted domain)**.** *An* expansion of the domain $S = \{l_i \mid i \in \mathbb{N}\}$ *of the lifted Σ_l-structure L is a set $D = \{d_i^j \mid \exists i, j \in \mathbb{N} : l_i \in S \wedge 1 \leqslant j \leqslant Mul^L(l_i)\}$ such that the d_i^j are distinct and $\forall i : d_i^1 = l_i$.*

We call d_i^1 a *base* element. We define a permutation on the set D having the cycles $(l_i, d_i^2 \ldots d_i^{Mul^L(l_i)})$:

Definition 23 (Permutation of the expanded domain)**.** *The permutation of an expansion D of a lifted domain is the function $\pi : D \mapsto D$ such that $\pi(d_i^j) = d_i^{j+1}$ for $1 \leqslant j < Mul^L(l_i) - 1$, and $\pi(d_i^{Mul^L(l_i)}) = d_i^1$.*

Notice that lifted domain elements with multiplicity zero have no image in D, and π is not defined on them.

The concrete interpretation of Σ-symbols in the expanded structure I is obtained by expanding the tuples in their lifted interpretations using π, according to

[2] Null multiplicities will prove useful for iterative methods in Section 4.4 on page 95.

Equations 4.19 to 4.21 on the preceding page. It follows from the correspondence between Definition 19 on page 71 and Equations 4.19 to 4.21 on the preceding page that π is an automorphism of I, and that the domain S of L is a backbone of π in I when I is actually a structure. The compression is lossless in the sense that, when the multiplicities in L are non-zero, the lifted structure (per π and S) of the expansion of L is L. However, I may not be a structure because the interpretation of a function symbol in I might not be a total function on its domain, as the following example shows.

Example 33. *Consider the lifted structure L with $S = \{a, b\}$, with types $A^L = \{a\}$, $B^L = \{b\}$, and for $f : A \times A \mapsto B$, $f^L = \{(a, a) \mapsto b\}$. Finally, let $mul(a) = mul(b) = 2$, The expanded domain is $\{a^1, a^2, b^1, b^2\}$ with permutation $(a^1a^2)(b^1b^2)$. The expansion of f^L is $\{(a^1, a^1) \mapsto b^1, (a^2, a^2) \mapsto b^2\}$. There is no entry for (a^1, a^2), so the expansion is not a total function.*

Consider now the lifted structure L with domain $S = \{a, b\}$, with type $A^L = \{a\}, B^L = \{b\}$, for $f : A \mapsto B$, $f^L = \{a \mapsto b\}$. Finally, $mul(a) = 1, mul(b) = 2$. The expanded domain is $\{a^1, b^1, b^2\}$ with permutation $(a^1)(b^1b^2)$. The expansion of f^L is $\{a^1 \mapsto b^1, a^1 \mapsto b^2\}$. This expansion gives two different values for a^1, so it is not a function.

To distinguish lifted structures that can be expanded into concrete ones from those that cannot, we introduce the notion of *regularity.*

Definition 24 (Regular function, regular lifted structure)**.** *The lifted interpretation of a function symbol is regular if its expansion defines a total function in the concrete domain, i.e., if it specifies exactly one image for every tuple in the concrete domain. A* regular lifted structure *is a lifted structure in which the interpretation of each function symbol is regular.*

We can now define the expansion of a regular lifted structure:

Definition 25 (Expansion of a regular lifted structure)**.** *Let L be a regular lifted structure over a lifted vocabulary. Then, I, the expansion of L, is the structure over the concrete vocabulary defined as follows:*

- *its domain is the union of each type expansion, i.e., $\bigcup_{T_i} exp_\pi(T_i^L)$;*
- *for each type symbol T, $T^I = exp_\pi(T^L)$;*
- *for each predicate symbol p, $p^I = exp_\pi(p^L)$;*
- *for each function symbol f, $f^I = exp_\pi(f^L)$.*

The expansion of a regular lifted structure does not involve any search. The time needed for this expansion is polynomial with respect to the size of the sets interpreting the types, and is generally negligible.

To further characterize regular functions, we introduce the concept of regular tuples of lifted elements.

Definition 26 (Expansion of a tuple)**.** *The* weak expansion *of a tuple* $\bar{l}$ *of lifted elements is its* π*-orbit:*

$$exp_\pi(\bar{l}) = o_\pi(\bar{l}) = \{\pi^i(\bar{l}) | i \in \mathbb{N}\} \tag{4.22}$$

The strong expansion *of tuple* $\bar{l}$ *is the cross-product of the expansion of its elements:*

$$sexp_\pi(\bar{l}) = \times_j \{\pi^i(l_j) | i \in \mathbb{N}\} \tag{4.23}$$

The weak and strong expansions of the empty tuple are the empty tuple.

If the expansion of one of the elements of the tuple is empty, both expansions are empty. Also, the weak expansion is a subset of the strong expansion.

Definition 27 (Regular tuple of lifted elements)**.** *A tuple of lifted elements is regular if and only if its weak expansion is equal to its strong expansion.*

We use $Mul(l_1, \ldots, l_n)$ (resp. $Lcm(l_1, \ldots, l_n)$) as a shorthand for the product of the multiplicities of lifted domain element l_i (resp. their least common multiple *lcm*). If tuple $\bar{l}$ is the empty tuple, $Mul(\bar{l}) = 1 = Lcm(\bar{l})$.

Proposition 5. *A tuple* $\bar{l}$ *is regular if and only if* $Mul(\bar{l}) = Lcm(\bar{l})$.

Proof. First, we observe that the size of the weak expansion (Definition 26) of a tuple $\bar{l}$ is finite: it is the order of the permutation defined by the cycles of its elements, i.e., $Lcm(l_1, \ldots, l_n)$.[3] Also, the weak expansion is a subset of the strong expansion, which has $Mul(l_1, \ldots, l_n)$ tuples. When those two numbers are equal, the two sets are equal. □

Example 34. *Let* (a, b) *be a tuple of two lifted domain elements with* $mul(a) = 2$ *and* $mul(b) = 4$*. Its weak expansion is* $\{(a^1, b^1), (a^2, b^2), (a^1, b^3), (a^2, b^4)\}$*, of size* $Lcm(2, 4) = 4$*. Note that, e.g., the tuple* (a^1, b^2) *does not belong to the weak expansion: thus,* (a, b) *is not regular. It is regular when* $mul(b) = 0$ *(the expansion of* b *and of* (a, b) *are empty), or when, e.g.,* $mul(b) = 3$*: in that case, the weak expansion is* $\{(a^1, b^1), (a^2, b^2), (a^1, b^3), (a^2, b^1), (a^1, b^2), (a^2, b^3)\}$*; it is identical to the strong expansion, of size* $Lcm(2, 3) = 6$*.*

[3] Taking the convention that the *lcm* of a tuple of numbers containing 0 is 0.

The expansion of two distinct tuples are disjoint (because the expansion of lifted elements are disjoint). Nullary and unary tuples of lifted elements are always regular. An n-ary tuple is regular when one of its elements has multiplicity zero, or every pair of its elements have multiplicities that are coprime.

Proposition 6 (Regular function)**.** *A function f^L is regular if, for all tuples $\bar{l}$ in the domain of f^L, it holds that (i) $\bar{l}$ is regular, and (ii) the multiplicity of $\bar{l}$ is a multiple of the multiplicity of its image.*

Proof. First, we show that the expansion of f^L gives at least one value for every tuple $\bar{d}$ in the concrete domain of f. Each element d_i of $\bar{d}$ is in the expansion of a lifted domain element $l_i \in T_i^L$. Tuple $\bar{l} = (l_1, \cdots, l_n) \in \bar{T}^L$ is in the lifted domain of f^L as f^L is total; it is regular by (i), hence $\bar{d}$ is in its weak expansion, and thus equal to $\pi^i(\bar{l})$ for some i. Then, $f^I(\bar{d})$ is in $exp_\pi(f^L(\bar{l}))$ because π is an automorphism.

Next, we show that the expansion of f^L gives at most one value for every tuple $\bar{d}$ in the concrete domain of f. Let $\bar{d}$ be in the weak expansion of $\bar{l}$, with $Lcm(\bar{l}) = m = Mul(\bar{l})$ by (i). We thus have $0 < m$. The expansion of f^L contains the pairs $\pi^i(\bar{l} \mapsto f^L(\bar{l}))$ and $\pi^{i+n\times m}(\bar{l} \mapsto f^L(\bar{l}))$, for any $\bar{l}$ in the domain of f^L, and for any i and n. The first element of these two pairs are identical by definition of m; the second elements are identical by (ii). □

We can now define the predefined functions and predicates supported by the method.

Definition 28 (Predefined predicates and functions)**.** *The predefined predicates are equality ($=$) and arithmetic comparisons ($<, \leqslant, \geqslant, >$). The predefined functions are the arithmetic operators ($+, -, \times, \div$).*

Note that disequality ($\neq$) and tuple equality are not predefined predicates.

Proposition 7. *The expansion of the interpretation of each predefined predicate (resp. function) in L is the interpretation of the predefined predicate (resp. function) in I.*

Proof. For example, the expansion of equality is, by equation 4.20 on page 73, $exp_\pi(=^L) = \{\pi^i(t, u) \mid (t = u)^L, i \in \mathbb{N}\}$, which is the intended interpretation of equality in the concrete domain. □

Example 35. *Consider a lifted structure L with a type T whose interpretation is $\{l, m\}$, with $mul(l) = 2 = mul(m)$. The lifted interpretation of the equality predicate is $\{(l, l), (m, m)\}$. If disequality were predefined, its lifted interpretation*

would be $\{(l,m),(m,l)\}$. *The expansion of* T^L *is* $\{l^1,l^2,m^1,m^2\}$. *The expansion of the interpretation of equality* $=$ *is* $\{(l^1,l^1),(l^2,l^2),(m^1,m^1),(m^2,m^2)\}$, *as expected. The expansion of the interpretation of disequality, however, would be* $\{(l^1,m^1),(l^2,m^2),(m^1,l^1),(m^2,l^2)\}$, *which is not the concrete interpretation of disequality: indeed,* $(l^1,l^2),(l^2,l^1),(m^1,m^2)$ *and* (m^2,m^1) *are missing. This motivates why disequality is not a predefined predicate. Note that the negation of the concrete interpretation of equality has the intended interpretation of disequality.*

Now consider a predicate for pair equality, with signature $T \times T \times T \times T \to \mathbb{B}$, *and with the intended interpretation that it is true when its first two arguments are equal to its last two arguments. Its interpretation in* L *would be :*

$$\{(l,l,l,l),(l,m,l,m),(m,m,m,m),(m,l,m,l)\}$$

The expansion of this interpretation would be:

$$\{(l^1,l^1,l^1,l^1),(l^2,l^2,l^2,l^2),(l^1,m^1,l^1,m^1),\ldots\}$$

which is not the intended interpretation in I*: for example,* (l^1,m^2,l^1,m^2) *is missing in the interpretation. This motivates why pair equality, and in general, tuple equality, is not a predefined predicate. Later, we'll use a tuple equality* $\bar{t} = \bar{u}$ *as a shorthand for a conjunction of equalities:* $\bigwedge_i t_i = u_i$.

4.3.2 Translation into a lifted sentence

We now present a translation of a sentence in FO(Type, Aggregate) into a sentence that allows domain compression, such that the translation is equisatisfiable with the original. [4] In this chapter, to simplify the presentation, we present the translation of a sentence in "negation normal form". Appendix B presents the transformation for any FO(Type, Aggregate) sentence.

The *negation normal form* of a formula ϕ is a formula logically equivalent to ϕ in which only atoms are negated. It can be obtained by syntactic transformation of the formula, per inductive rules defined in Table 4.1 on the next page. The first column shows the possible syntactical forms of the formula; the second column defines the transformations of the formula; the third column defines the transformation of its negation;

The translation $\overline{\chi(\phi)}$ of a sentence ϕ is the conjunction of

(A) sentences expressing the regularity of each function symbol,

[4] The translation of a theory, i.e., of a set of sentences, is the translation of the conjunction of the sentences.

Formula ϕ	$NF(\phi)$	$NF(\neg\phi)$
true	**true**	**false**
$p()$	$p()$	$\neg p()$
$p(\bar{t})$	$p(NF(\bar{t}))$	$\neg p(NF(\bar{t}))$
$\psi_1 \wedge \psi_2$	$NF(\psi_1) \wedge NF(\psi_2)$	$NF(\neg\psi_1) \vee NF(\neg\psi_2)$
$\psi_1 \vee \psi_2$	$NF(\psi_1) \vee NF(\psi_2)$	$NF(\neg\psi_1) \wedge NF(\neg\psi_2)$
$\neg\psi$	(see third column)	$NF(\psi)$
$T \triangleq \{t_1, \ldots, t_n\}$	$T \triangleq \{t_1, \ldots, t_n\}$	
$\forall \bar{x} \in \bar{T} : \psi$	$\forall \bar{x} \in \bar{T} : NF(\psi)$	$\exists \bar{x} \in \bar{T} : NF(\neg\psi)$
$\exists \bar{x} \in \bar{T} : \psi$	$\exists \bar{x} \in \bar{T} : NF(\psi)$	$\forall \bar{x} \in \bar{T} : NF(\neg\psi)$
Term t	$NF(t)$	
x	x	
$c()$	$c()$	
$f(\bar{t})$	$f(NF(\bar{t}))$	
$\sum_{\bar{x} \in \bar{T} : \psi} t_0$	$\sum_{\bar{x} \in \bar{T} : NF(\psi)} NF(t_0)$	

Table 4.1: Negation normal form of formulas and terms

(B) the transformation W of the sentence in negation normal form, $W(NF(\phi))$.

For part (A) of the translation, for each function symbol $f{:}\bar{T}{\to}T$, the following sentences are used to ensure that the function interpretations are regular (Proposition 6 on page 76):

$$\forall \bar{x} \in \bar{T}^0 : Mul(\bar{x}) = Lcm(\bar{x}) \tag{4.24}$$

$$\forall \bar{x} \in \bar{T}^0 : \exists n \in \mathbb{N} : Mul(\bar{x}) = n \times mul(f(\bar{x})) \tag{4.25}$$

where T^0 is the set of elements of type T whose multiplicity is not zero.[5]

For a constant $c : () \to T$, Equation (4.25) reduces to:

$$mul(c()) = 1 \tag{4.26}$$

The inductive definition for the transformation W (for Part B) is shown in Table 4.2 on page 80. This definition uses auxiliary functions S, w, s and χ which are also defined in the Table. W and S translate formulas; χ translates

[5] $Mul(\bar{x})$ (resp. $Lcm(\bar{x})$) stands for the product (resp. the *lcm*) of the multiplicities of the elements of the lifted tuple assigned to $\bar{x}$.

terms; w, s and *Reg* produce a formula from a (tuple of) term(s) which is added as a conjunct to the translation of the atom containing the term(s). The leftmost column of Table 4.2 on the following page shows the possible syntactical forms in the concrete sentence; for formulas and terms, the second and third columns define the W, S, w, s and χ functions inductively; terms have an extra set of rows that define the function χ.

We'll show in the next section that, for any formula ϕ, if $W(\phi)$ (resp. $S(\phi)$) is true in $L[\bar{y} : \bar{l}]$, then ϕ is true in any $I[\bar{y} : \bar{d}]$ where I is the expansion of L and $\bar{d}$ is a weak (resp. strong) instance of $\bar{l}$. Functions w and s play similar roles for terms.

The transformation of a sentence essentially consists of

- adding a regularity atom $w(\bar{t})$ (resp. $Reg(\bar{t})$) to the translation of an atom $p(\bar{t})$ (Rule 4.29) to ensure that, if the lifted atom is true in $L[\bar{y} : \bar{l}]$, then the original atom is true in every weak (resp. strong) instance of $L[\bar{y} : \bar{l}]$;
- using $\bar{x} \in \bar{T}^0$ instead of $\bar{x} \in \bar{T}$ (i.e., adding a $0 < Mul(\bar{x})$ filter), to cope with lifted domains elements with an empty expansion;
- multiplying the term t_0 in an aggregate term by some decompression factor: in Rule (4.39), it is the number $Mul(\bar{x})$ of possible concrete variable assignments; in Rule (4.42), it is that number multiplied by the fraction of concrete assignments that make $p(\bar{t}, \bar{u})$ true.

Note that p (resp. f) occurring in the syntactical forms (leftmost column) matches all predicate (resp. function) symbols in the vocabulary of sentence ϕ, including the predefined ones (Definition 28 on page 76).

Note also that Rule (4.33) assumes that the identifiers $t_1, \ldots, t_n$ are not used elsewhere in the sentence to be satisfied. In that case, the formula $T \triangleq \{t_1, \ldots, t_n\}$ is equivalent to simply $\#\{x \in T\} = n$. If some identifiers, e.g., $t_1, \ldots, t_m$, are used elsewhere in the sentence to be satisfied, then the interpretation of the type in the lifted model must have domain elements identified by those identifiers, and those elements must have a multiplicity of one. The translation of $T \triangleq \{t_1, \ldots, t_m, \ldots, t_n\}$ is then:

$$T \triangleq \{t_1, \ldots, t_m, \ldots\} \wedge \sum_{x \in T} mul(x) = n \wedge \bigwedge_{1 < i < m} mul(t_i) = 1 \qquad (4.40)$$

where $T \triangleq \{t_1, \ldots, t_m, \ldots\}$ indicates that the list of lifted identifiers of type T is not fully known but must include $\{t_1, \ldots, t_m\}$.

Table 4.3 shows transformation rules for particular syntactical forms. Generally, it is beneficial to do equivalence-preserving transformations of sentences to

Formula ϕ	$W(\phi)$	$S(\phi)$	
true/**false**	**true**/**false**	**true**/**false**	(4.27)
$p()$	$p()$	$p()$	(4.28)
$p(\bar{t})$	$p(\chi(\bar{t})) \wedge w(\bar{t})$	$p(\chi(\bar{t})) \wedge Reg(\bar{t})$	(4.29)
$\neg p(\bar{t})$	$\neg p(\chi(\bar{t})) \wedge w(\bar{t})$	$\neg p(\chi(\bar{t})) \wedge s(\bar{t})$	(4.30)
$\psi_1 \wedge \psi_2$	$W(\psi_1) \wedge W(\psi_2)$	$S(\psi_1) \wedge S(\psi_2)$	(4.31)
$\psi_1 \vee \psi_2$	$W(\psi_1) \vee W(\psi_2)$	$S(\psi_1) \vee S(\psi_2)$	(4.32)
$T \triangleq \{t_1, \ldots, t_n\}$	$\sum_{x \in T} mul(x) = n$	$\sum_{x \in T} mul(x) = n$	(4.33)
$\forall \bar{x} \in \bar{T} : \psi$	$\forall \bar{x} \in \bar{T}^0 : S(\psi)$	$\forall \bar{x} \in \bar{T}^0 : S(\psi)$	(4.34)
$\exists \bar{x} \in \bar{T} : \psi$	$\exists \bar{x} \in \bar{T}^0 : W(\psi)$	$\exists \bar{x} \in \bar{T}^0 : S(\psi)$ $\vee [Reg(\bar{y}) \wedge W(\psi)]$	(4.35)
Term t	$w(t)$	$s(t)$	
x	**true**	**true**	(4.36)
$c()$	**true**	**true**	(4.37)
$f(\bar{t})$	$w(\bar{t})$	$s(\bar{t})$	(4.38)
$\sum_{\bar{x} \in \bar{T} : \psi} t_0$	$\forall \bar{x} \in \bar{T}^0 : s(t_0)$ $\wedge [S(\psi) \vee S(NF(\neg\psi))]$	$\forall \bar{x} \in \bar{T}^0 : s(t_0)$ $\wedge [S(\psi) \vee S(NF(\neg\psi))]$	(4.39)
Term t	$\chi(t)$		
x	x		
$c()$	$c()$		
$f(\bar{t})$	$f(\chi(\bar{t}))$		
$\sum_{\bar{x} \in \bar{T} : \psi} t_0$	$\sum_{\bar{x} \in \bar{T}^0 : S(\psi)} Mul(\bar{x}) \times \chi(t_0)$		

– $w(\bar{t})$ is a shorthand for $\bigwedge_i w(t_i)$ (and similarly for the s function).
– $\bar{y}$ is a shorthand for $vars(\phi)$.
– $Reg(\bar{t})$ is defined as $s(\bar{t}) \wedge (Mul(\chi(\bar{t}))=Lcm(\chi(\bar{t})) \vee Mul(vars(\bar{t}))=Lcm(vars(\bar{t})))$.
– $Reg(\bar{y})$ simplifies to $Mul(\bar{y}) = Lcm(\bar{y})$.
– $\bar{T}^0$ denotes the subset of tuples in $\bar{T}$ whose multiplicity is non-zero.

Table 4.2: Transformation of formulas and terms in negation normal form

Special rules		
$\exists \bar{x} \in \bar{T} : P(\bar{\bar{t}}, \bar{\bar{u}}) \wedge \psi$	$S(\phi) = \exists \bar{x} \in \bar{T}^0 : S(\psi) \vee [Reg(\bar{y}) \wedge W(\psi)] \vee [Reg(\bar{\bar{u}}) \wedge W(P(\chi(\bar{\bar{t}}, \bar{\bar{u}}))) \wedge S(\psi)]$	(4.41)
$\sum_{\bar{x} \in \bar{T} : p(\bar{t}, \bar{u}) \wedge \psi} t_0$	$\chi(t) = \sum_{\bar{x} \in \bar{T}^0 : p(\chi(\bar{t}, \bar{u})) \wedge S(\psi)} \frac{Lcm(\chi(\bar{t}, \bar{u})) \times Mul(\bar{x})}{Lcm(\chi(\bar{t})) \times Lcm(\chi(\bar{u}))} \times \chi(t_0)$	(4.42)
	$w(t) = \forall \bar{x} \in \bar{T}^0 : Reg(\bar{t}) \wedge w(\bar{u}) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$	
	$s(t) = \forall \bar{x} \in \bar{T}^0 : Reg(\bar{t}) \wedge Reg(\bar{u}) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$	
$(\sum_{\bar{x} \in \bar{T} :} t_0) \in \{0, n\}$	$S(\phi) = (\sum_{\bar{x} \in \bar{T}^0 : p(\chi(\bar{t}, \bar{u})) \wedge S(\psi)} \frac{Lcm(\chi(\bar{t}, \bar{u})) \times Mul(\bar{x})}{Lcm(\chi(\bar{t})) \times Lcm(\chi(\bar{u}))} \times \chi(t_0)) \in \{0, n\}$	(4.43)
$p(\bar{t}, \bar{u}) \wedge \psi$	$\wedge \forall \bar{x} \in \bar{T}^0 : Reg(\bar{t}) \wedge w(\bar{u}) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$	
$\sum_{\bar{x} \in \bar{T} : \bar{t} = \bar{u} \wedge \psi} t_0$	$\chi(t) = \sum_{\bar{x} \in \bar{T}^0 : \chi(\bar{t}) = \chi(\bar{u}) \wedge S(\psi)} \frac{Mul(\bar{x})}{Lcm(\chi(\bar{u}))} \times \chi(t_0)$	(4.44)
	$s(t) = \forall \bar{x} \in \bar{T}^0 : (Reg(\bar{t}) \vee Reg(\bar{u})) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$	
$(\sum_{\bar{x} \in \bar{T} :} t_0) \in \{0, n\}$	$S(\phi) = (\sum_{\bar{x} \in \bar{T}^0 : \chi(\bar{t}) = \chi(\bar{u}) \wedge S(\psi)} \frac{Mul(\bar{x})}{Lcm(\chi(\bar{u}))} \times \chi(t_0)) \in \{0, n\}$	(4.45)
$\bar{t} = \bar{u} \wedge \psi$	$\wedge \forall \bar{x} \in \bar{T}^0 : (Reg(\bar{t}) \vee Reg(\bar{u})) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$	

– $\bar{\bar{t}}$ is a shorthand for the concatenation of tuples $\overline{t_i}$, i.e., $(\overline{t_1}, \ldots, \overline{t_n})$.
– $P(\bar{\bar{t}}, \bar{\bar{u}})$ is a shorthand for $\bigwedge_i p_i(\overline{t_i}, \overline{u_i})$. $\bar{t} = \bar{u}$ is a shorthand for $\bigwedge_i t_i = u_i$.
– The special rules are applied only when $vars(\bar{\bar{t}}) \subseteq \{\bar{x}\}$ and $vars(\bar{\bar{u}}) \cap \{\bar{x}\} = \varnothing$ (or the same with $\bar{t}, \bar{u}$ depending on context)
– $t \in \{0, n\}$ is a shorthand for $t = 0 \vee t = n$ where n is an integer.

Table 4.3: Special rules for the transformation of sentences

obtain sentences of the form allowing application of the special rules because these special rules afford better compression of the domain. Indeed, they weaken the regularity condition $Reg(\bar{\bar{t}}, \bar{\bar{u}})$ (required by Rule 4.29) for the atom $P(\bar{\bar{t}}, \bar{\bar{u}})$ in Rule (4.41), the regularity condition $Reg(\bar{t}, \bar{u})$ for the atom $p(\bar{t}, \bar{u})$ in Rules (4.42) and (4.43), and the regularity condition $\bigwedge_i Reg(t_i, u_i)$ for Rules (4.44) and (4.45). Note the subtle differences in the filter conditions of the quantifications and aggregates in the special rules: the filter in Rule (4.41) has a conjunction of predicates $P(\bar{\bar{t}}, \bar{\bar{u}})$; the filters in Rules (4.42) and (4.43) have a single predicate $p(\bar{t}, \bar{u})$, and the filters in Rules (4.44) and (4.45) have a conjunction of equalities $\bar{t} = \bar{u}$.

Example 36. *The sentence "Assign each pigeon to pigeonhole(s) such that each hole has exactly 2 pigeons" is:*

$$\forall h \in Hole : \#\{p \in Pigeon \mid isIn(p, h)\} = 2$$

Its transformation by Rule (4.42) (with $\phi = \textbf{\textit{true}}$) and 4.34 is:[6]

$$\forall h \in Hole : 0 < mul(h) \Rightarrow \tag{4.46}$$

$$\sum_{p \in Pigeon} (\frac{Lcm(p,h)}{mul(h)} \text{ if } 0 < mul(p) \wedge isIn(p,h)) = 2$$

If the general rule (4.39) were used instead of (4.42), then the factor in the translation of the sum aggregate would be $mul(p)$. However, rule (4.29) would add the constraint that the lifted tuples (p,h) be regular when $isIn(p,h)$ is true, i.e., that $Lcm(p,h) = mul(p) \times mul(h)$. Then, the multiplicity of p could not be the double of the multiplicity of h unless $mul(h)$ is 1, so the compression would be severely restrained.

4.3.3 Equisatisfiability

Theorem 1 (Equisatisfiability)**.** *An FO(Type, Aggregate) sentence has a model I if and only if its translation has a model L.*

We call the FO(Type, Aggregate) sentence, the "original" sentence, and its translation, the "lifted" sentence.

Proving the forward implication is trivial: if I is a model of the original sentence, then the lifted structure L which extends I with multiplicity functions that always evaluate to 1 is a model of the translated sentence: indeed, the added constraints (A) are trivially satisfied, and the transformed sentence (B) is equivalent to the original one.

Proving the converse (i.e., that, if L is a model of the translated sentence, then its expansion I per Section 4.3.1 on page 70 is a model of the original sentence with the given number of concrete elements) is long and complex. First, the expansion of L is possible because L is a regular structure: its functions are regular (part (A) of the translated sentence). Secondly, we prove that I is a model of the concrete sentence. In what follows, we provide more details of that proof.

We use l for lifted domain elements from L and d for domain elements from I. We begin with a definition.

Definition 29 (Expansion of a variable assignment)**.** *The weak (resp. strong) expansion of a lifted variable assignment $[\bar{x} : \bar{l}]$ is the set of concrete variable assignments $[\bar{x} : \bar{d}]$ where $\bar{d}$ is in the weak (resp. strong) expansion of tuple $\bar{l}$.*

[6] Recall that a cardinality aggregate is a shorthand for a sum aggregate whose term is 1.

We say that assignment $[\bar{x} : \bar{d}]$ is a *weak instance* (resp. *strong instance*) of $[\bar{x} : \bar{l}]$, and structure $I[\bar{x} : \bar{d}]$ is a *weak instance* (resp. *strong instance*) of $L[\bar{x} : \bar{l}]$. Note that there are $Mul(\bar{x})$ strong instances of $[\bar{x} : \bar{l}]$, but only $Lcm(\bar{x})$ weak instances of $[\bar{x} : \bar{l}]$. Thanks to the $0 < Mul(\bar{x})$ filters in the transformation of quantified formulas and aggregates, we consider only the lifted variable assignments that have instances.

The theorem is a direct consequence of the following four related properties:

Proposition 8 (Weak preservation of truth values)**.** *If $W(\psi)$ is true in $L[\bar{y} : \bar{l}]$, then, ψ is true in every weak instance of $L[\bar{y} : \bar{l}]$: for any i, ψ is true in $I[\bar{y} : \pi^i(\bar{l})]$.*

Proposition 9 (Strong preservation of truth values)**.** *If $S(\psi)$ is true in $L[\bar{y} : \bar{l}]$, then ψ is true in every strong instance of $L[\bar{y} : \bar{l}]$.*

Proposition 10 (Weak preservation of values)**.** *If $w(t)$ is true in $L[\bar{y} : \bar{l}]$, then, for any i, $t^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\chi(t)^{L[\bar{y}:\bar{l}]})$.*

Proposition 11 (Strong preservation of values)**.** *If $s(t)$ is true in $L[\bar{y} : \bar{l}]$, then, for any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$, there exists an i such that $t^{I[\bar{y}:\bar{d}]} = \pi^i(\chi(t)^{L[\bar{y}:\bar{l}]})$.*

Proposition 8 applied to a sentence (without variables) implies our goal: if $W(NF(\phi))$ is true in L, then $NF(\phi)$, and thus ϕ, is true in the expansion I of L.

The proof of these four properties is done by structural induction over the syntactic tree of the original sentence. The base cases are the leaves of the syntactic tree; for the inductive step, we prove that the consequence of a property holds in a node of the syntactic tree under the assumption that the precondition of the property holds and under the induction hypothesis that the properties above hold in all the branches of the node. We consider every language construct and their translation per Table 4.2.

For brevity, in the proof, we'll often use L' to denote a structure extended with a variable assignment $[\bar{y} : \bar{l}]$. When we say that an expression $\chi(e)$ is evaluated in L', and it is in the scope of quantifiers or sum aggregates that collectively declare variables $\bar{y}$, we mean that it is evaluated in $L[\bar{y} : \bar{l}]$. Similarly, we'll often use I' to denote a (weak or strong, depending on context) instance of L'. When we say that e is evaluated in I', we mean that it is evaluated in $I[\bar{y} : \bar{d}]$, with $[\bar{y} : \bar{d}]$ a (weak or strong) instance of $[\bar{y} : \bar{l}]$. Because quantifications and aggregations in the definition of W, S, w, w, χ are over $\bar{T}^0$, we only consider lifted variable assignments that have instances.

Lemmas

Lemma 1 (Weak preservation of tuples)**.** *If $w(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$, then for any i, $\bar{t}^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\chi(\bar{t})^{L[\bar{y}:\bar{l}]})$.*

Proof. Since t_j is a sub-term of $\bar{t}$, the induction hypothesis holds for each t_j: the weak preservation of the value of a term (Proposition 10 on the previous page) holds for each t_j in $\bar{t}$. Its precondition, i.e., $w(t_j)$ holds in L'. By consequence, for any i, $t_j^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\chi(t_j)^{L'})$. Thus, for any i, $\bar{t}^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\chi(\bar{t})^{L'})$. □

Lemma 2 (Strong preservation of tuples)**.** *If $Reg(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$ (where $vars(\bar{t}) \subseteq \{\bar{y}\}$), then, for any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$, there exists an i such that $\bar{t}^{I[\bar{y}:\bar{d}]} = \pi^i(\chi(\bar{t})^{L[\bar{y}:\bar{l}]})$.*

Proof. According to Table 4.2, $Reg(\bar{t})$ is true when $s(t_j)$ is true in L' for any t_j in $\bar{t}$ and either $Mul(\chi(\bar{t})) = Lcm(\chi(\bar{t}))$ or $Mul(vars(\bar{t})) = Lcm(vars(\bar{t}))$ is true in L'.

Since t_j is a sub-term of $\bar{t}$, the induction hypothesis holds for each t_j: the strong preservation of the value of a term (Proposition 11 on the preceding page) holds for each t_j in $\bar{t}$. Its precondition holds ($s(t_j)$ is true): as a consequence, for any t_j and I', there is an i_j such that $t_j^{I'} = \pi^{i_j}(\chi(t_j)^{L'})$, i.e., $t_j^{I'}$ is in the expansion of $\chi(t_j)^{L'}$.

When the first equality $Mul(\chi(\bar{t})) = Lcm(\chi(\bar{t}))$ holds, $\chi(\bar{t})$ is regular in L', and its weak and strong expansion are identical. Any element $\bar{t}^{I'}$ in the strong expansion of $\chi(\bar{t})^{L'}$ is necessarily of the form $\pi^i(\chi(\bar{t})^{L'})$ for some i.

Let's denote $vars(\bar{t})$ by $\bar{y}_1$, and the other variables in $\bar{y}$ by $\bar{y}_2$. In other words, $\bar{y} = \bar{y}_1 :: \bar{y}_2$. Similarly, $\bar{l} = \bar{l}_1 :: \bar{l}_2$, and $\bar{d} = \bar{d}_1 :: \bar{d}_2$. When the second equality, $Mul(vars(\bar{t})) = Lcm(vars(\bar{t}))$, holds, i.e., $Mul(\bar{y}_1) = Lcm(\bar{y}_1)$, $\bar{d}_1 = \pi^i(\bar{l}_1)$ for some i. Because $s(\bar{t})$ implies $w(\bar{t})$, the precondition for Lemma 1 holds: by consequence, for any $\bar{d}$, $\bar{t}^{I[\bar{y}:\bar{d}]} = \bar{t}^{I[\bar{y}_1:\bar{d}_1]} = \bar{t}^{I[\bar{y}_1:\pi^i(\bar{l}_1)]} = \pi^i(\chi(t)^{L[\bar{y}_1:\bar{l}_1]}) = \pi^i(\chi(t)^{L[\bar{y}:\bar{l}]})$. □

Base cases

Rule for $p()$ (Rule 4.28 on page 80) If $\chi(p())$ (i.e., $p()$) is true in L', then, $p()$ is true in any I' (including $I[\bar{x} : \pi^i(\bar{l})]$ for any i), by construction of p^I.

Rule for x (Rule 4.36 on page 80)

For weak preservation (Proposition 10 on page 83): for any i, $x^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(l_x) = \pi^i(\chi(x)^{L'})$, where l_x is the value assigned to x by $[\bar{y} : \bar{l}]$.

For strong preservation (Proposition 11 on page 83): the value of x in any I' is in the expansion of l_x. This expansion forms a cycle of π, hence, for any I', there is an i such that $x^{I'} = \pi^i(l_x) = \pi^i(\chi(x)^{L'})$.

Rule for $c()$ (Rule 4.37 on page 80) For weak preservation (Proposition 10 on page 83): for any i, $c()^{I[\bar{y}:\pi^i(\bar{l})]} = c()^I = c()^L = \chi(c())^L = \pi^i(\chi(c())^{L'})$, by construction of the interpretation of c in I and because the multiplicity of $c()$ is 1 in L'.

For strong preservation (Proposition 11 on page 83): similarly, for any I', and for any i, $c()^{I'} = c()^I = c()^L = \pi^i(\chi(c()))^{L'}$.

Rule for $T \triangleq \{t_1, \ldots, t_n\}$ (Rule 4.33)

If identifiers $t_1, \ldots, t_n$ are not used elsewhere in the sentence to be satisfied, then the sentence $T \triangleq \{t_1, \ldots, t_n\}$ is equivalent to simply $\#\{x \in T \mid \mathbf{true}\} = n$, and its translation is described by Rule 4.39 (see proof below).

Predicate application Rule for $p(\bar{t})$ (Rule 4.29 on page 80)

(weak) *If $p(\chi(\bar{t})) \wedge w(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$, then, for any i, $p(\bar{t})$ is true in $I[\bar{y} : \pi^i(\bar{l})]$.*

Proof. By assumption, $w(\bar{t})$ is true in L'. By lemma 1 and by definition of the expansion of p^L (Equation 4.20), for any i, $p(\bar{t})^{I[\bar{y}:\pi^i(\bar{l})]} = p^I(\bar{t}^{I[\bar{y}:\pi^i(\bar{l})]}) = p^I(\pi^i(\chi(\bar{t})))^{L'} = p(\chi(\bar{t}))^{L'}$ by definition of the expansion of p^L. Since $p(\chi(\bar{t}))$ is true in L' by assumption, $p(\bar{t})$ is true in $I[\bar{y} : \pi^i(\bar{l})]$. □

(strong) *If $p(\chi(\bar{t})) \wedge Reg(\bar{t})$ is true in $L' = L[\bar{y} : \bar{l}]$, then, $p(\bar{t})$ is true in every strong instance of L'.*

Proof. By assumption, $Reg(\bar{t})$ is true in L'. By Lemma 2, there exists an i such that $\bar{t}^{I[\bar{y}:\bar{d}]} = \pi^i(\chi(\bar{t})^{L'})$. Again, we have $p(\bar{t})^{I[\bar{y}:\bar{d}]} = p^I(\bar{t}^{I[\bar{y}:\pi^i(\bar{l})]}) = p^I(\pi^i(\chi(\bar{t})^{L'})) = p(\chi(\bar{t}))^{L'}$. Since $p(\chi(\bar{t}))$ is true in L' by assumption, $p(\bar{t})$ is true in $I[\bar{y} : \bar{d}]$ and strong preservation holds. □

Rule for $\neg p(\bar{t})$ (Rule 4.30 on page 80)

(weak) *If $\neg p(\chi(\bar{t})) \wedge w(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$, then, for any i, $\neg p(\bar{t})$ is true in $I[\bar{y} : \pi^i(\bar{l})]$.*

Proof. By assumption, $\neg p(\chi(\bar{t}))$ and $w(\bar{t})$ are true in L'. So, we can apply Lemma 1 on $\bar{t}$: for any i, $t^{I[\bar{x}:\pi^i(\bar{l})]} = \pi^i(tr(t)^{L[\bar{x}:\bar{l}]}$. So $p(\bar{t})^{I[\bar{x}:\pi^i(\bar{l})]}$ has the same truth value as $p(\chi(\bar{t}))^{L[\bar{x}:\bar{L}]}$. This holds also for their negation, hence weak preservation holds. □

(strong) *If* $\neg p(\chi(\bar{t})) \wedge s(\bar{t})$ *is true in* $L' = L[\bar{y} : \bar{l}]$*, then,* $\neg p(\bar{t})$ *is true in every strong instance of* L'*.*

Proof. By assumption, $\neg p(\chi(\bar{t}))$ and $s(\bar{t})$ are true in L'. By the induction hypothesis, Proposition 11 holds. Its precondition, $s(t_j)$, is true in L' for any t_j in $\bar{t}$: hence, $t_j^{I'}$ is in the expansion of $\chi(t_j)^{L'}$ for any I', and $\bar{t}^{I'}$ is in the cross-product of the expansions of each $\chi(t_j)^{L'}$, for any I'. By assumption that $p(\chi(\bar{t}))$ is false, and by construction of p^I (Definition 25 on page 74 and Equation 4.20 on page 73), p^I is false on all elements of this cross-product, and thus, $p(\bar{t})$ is false in any I'. □

Logic connectives

Rule for $\psi_1 \wedge \psi_2$ (Rule 4.31 on page 80)

(weak) *If* $W(\psi_1) \wedge W(\psi_2)$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any* i*,* $\psi_1 \wedge \psi_2$ *is true in* $I[\bar{y} : \pi^i(\bar{l})]$*.*

Proof. By the induction hypothesis, Proposition 8 on page 83 holds for both ψ_j. The precondition, i.e., $W(\psi_j)$, also holds for both j by assumption. By consequence, both ψ_j are true in $I[\bar{y} : \pi^i(\bar{l})]$ for any i, and thus $\psi_1 \wedge \psi_2$ too. □

(strong) *If* $S(\psi_1) \wedge S(\psi_2)$ *is true in* $L' = L[\bar{y} : \bar{l}]$*, then,* $\psi_1 \wedge \psi_2$ *is true in every strong instance of* L'*.*

Proof. By the induction hypothesis, Proposition 8 on page 83 holds for both ψ_j. The precondition, i.e., $S(\psi_j)$, also holds for both j by assumption. By consequence, both ψ_j are true in any strong instance of L', and thus $\psi_1 \wedge \psi_2$ too. □

Rule for $\chi(\psi_1 \vee \psi_2)$ (Rule 4.32 on page 80) A similar reasoning can be used to prove the property for a disjunction.

Quantification

Rule for $\forall \bar{x} \in \bar{T} : \psi$ (Rule 4.34 on page 80)

Let's call $\bar{y}$ the free variables of ϕ, i.e., of $\forall \bar{x} \in \bar{T} : \psi$.

(strong) *If* $\forall \bar{x} \in \bar{T} : S(\psi)$ *is true in* $L' = L[\bar{y} : \bar{l}]$*, then,* $\forall \bar{x} \in \bar{T} : \psi$ *is true in every strong instance of* L'*.*

Proof. By assumption, for all $\bar{l}_x$ in $\bar{T}^L$, $S(\psi)$ is true in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. By the induction hypothesis, strong preservation (Proposition 9 on page 83) holds for ψ: since its precondition $S(\psi)$ is satisfied in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$, ψ is true in any strong instance of $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. Hence, ϕ is in true in any strong instance of $L[\bar{y} : \bar{l}]$. □

(weak) *If* $\forall \bar{x} \in \bar{T} : S(\psi)$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any* i*,* $\forall \bar{x} \in \bar{T} : \psi$ *is true in* $I[\bar{y} : \pi^i(\bar{l})]$.

Proof. The regularity condition for strong preservation of the truth value of $\forall \bar{x} \in \bar{T} : \psi$ (Proposition 9 on page 83) is satisfied by assumption: thus, ϕ is true in any instance I' of $L[\bar{y} : \bar{l}]$, and thus in particular in $I[\bar{y} : \pi^i(\bar{l})]$. □

Rule for $\exists \bar{x} \in \bar{T} : \psi$ (Rule 4.35 on page 80)

(weak) *If* $\exists \bar{x} \in \bar{T} : W(\psi)$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any* i*,* $\exists \bar{x} \in \bar{T} : \psi$ *is true in* $I[\bar{y} : \pi^i(\bar{l})]$.

Proof. By assumption, there is a $\bar{l}_x$ in $\bar{T}^L$ such that $W(\psi)$ is true in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. By the induction hypothesis, weak preservation (Proposition 8 on page 83) holds for ψ, and by assumption, its precondition $W(\psi)$ is satisfied in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$: by consequence, for any i, ψ is true in $I[\bar{x} : \pi^i(\bar{l}_x) :: \bar{y} : \pi^i(\bar{l})]$. Hence, ϕ is in true in $I[\bar{y} : \pi^i(\bar{l})]$ for any i. □

(strong) *If* $\exists \bar{x} \in \bar{T} : S(\psi) \vee (Reg(\bar{y}) \wedge W(\psi))$ *is true in* $L' = L[\bar{y} : \bar{l}]$*, then,* $\exists \bar{x} \in \bar{T} : \psi$ *is true in every strong instance of* L'*.*

Proof. By assumption, there is a $\bar{l}_x$ in $\bar{T}^L$ such that (i) $S(\psi)$ is true in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$, or (ii) the free variables of ϕ are regular and $W(\psi)$ is true in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. In the first case, by the induction hypothesis, strong preservation (Proposition 9 on page 83) holds for ψ, and by assumption, its precondition $S(\psi)$ holds in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$: by consequence, ψ is true in every strong

instance of $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$, and ϕ is true in every strong instance of $L[\bar{y} : \bar{l}]$. In the second case, $\bar{l}$ is regular, and thus, for any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$, there is an i such that $\bar{d} = \pi^i(\bar{l})$. By the induction hypothesis, weak preservation (Proposition 8 on page 83) holds for ψ, and, by assumption, its precondition $W(\psi)$ is satisfied in $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. by consequence, ψ is true in $I[\bar{x} : \pi^i(\bar{l}_x) :: \bar{y} : \pi^i(\bar{l})]$, and thus ϕ is true in $I[\bar{y} : \pi^i(\bar{l})] = I[\bar{y} : \bar{d}]$. □

Function application **Rule for $f(\bar{t})$ (Rule 4.38 on page 80)**

(weak) *If $w(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$, then, for any i, $f(\bar{t})^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\chi(f(\bar{t}))^{L[\bar{y}:\bar{l}]})$.*

Proof. By assumption, $w(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$. By Lemma 1 on page 84: for any i, $\bar{t}^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\chi(\bar{t})^{L'})$. Since π is an automorphism, for any i, $f(\bar{t}^{I[\bar{y}:\pi^i(\bar{l})]}) = f(\pi^i(\chi(\bar{t})^{L'})) = \pi^i(\chi(f(\bar{t}))^{L'})$. □

(strong) *If $s(\bar{t})$ is true in $L[\bar{y} : \bar{l}]$, then, for any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$, there exists an i such that $f(\bar{t})^{I[\bar{y}:\bar{d}]} = \pi^i(\chi(f(\bar{t}))^{L[\bar{y}:\bar{l}]})$.*

Proof. By the induction hypothesis, the strong preservation of the terms in $\bar{t}$ holds. By assumption, the precondition $s(t_j)$ holds for each term in $\bar{t}$. By consequence, for any strong instance $I' = I[\bar{y} : \bar{d}]$ of $L' = L[\bar{y} : \bar{l}]$, there is an i_j such that $t_j^{I'} = \pi^{i_j}(\chi(t_j)^{L'})$. Hence, $\bar{t}^{I'}$ is in the cross-product of the expansions of the $\chi(t_j)^{L'}$. Because f is regular, the tuples in its domain are regular, and in particular, $\chi(\bar{t})^{L'}$ is regular. Hence, there is an i such that $\bar{t}^{I'} = \pi^i(\chi(\bar{t})^{L'})$. Since π is an automorphism, $f(\bar{t})^{I'} = f(\pi^i(\chi(\bar{t})^{L'})) = \pi^i(f(\chi(\bar{t}))^{L'}) = \pi^i(\chi(f(\bar{t}))^{L'})$. □

Sum aggregate
Rule for $\sum_{\bar{x}\in\bar{T}:\psi} t_0$ (Rule 4.39 on page 80)

Recall that $\bar{y}$ are the free variables of t, i.e., of $\sum_{\bar{x}\in\bar{T}}(t_0 \textit{ if } \psi)$, and that, for any numeric term t in any lifted structure L', $\pi^i(t^{L'}) = t^{L'}$.

(strong) *If $\forall \bar{x} \in \bar{T}^0 : s(t_0) \wedge Reg(\bar{y}) \wedge [S(\psi) \vee S(NF(\neg\psi))]$ is true in $L[\bar{y} : \bar{l}]$, then, for any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$, there exists an i such that $\sum_{\bar{x}\in\bar{T}:\psi} t_0{}^{I[\bar{y}:\bar{d}]} = \pi^i(\sum_{\bar{x}\in\bar{T}}(Mul(l_x) \times \chi(t_0) \textit{ if } \psi)^{L[\bar{y}:\bar{l}]})$.*

Proof. Indeed, $t^{I[\bar{y}:\bar{d}]}$ is the sum of $(t_0 \textit{ if } \psi \textit{ else } 0)^{I[\bar{x}:\bar{d}_x::\bar{y}:\bar{d}]}$ for each strong instance $[\bar{x} : \bar{d}_x]$ of $[\bar{x} : \bar{l}_x]$ in any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$. By the

induction hypothesis, the property of strong preservation (Property 11 on page 83) holds for t_0. Its precondition, $s(t_0)$ holds by assumption, and t_0 is a number with multiplicity 1: by consequence, $t_0^{I[\bar{x}:\bar{d}_x::\bar{y}:\bar{d}]} = \chi(t_0)^{L[\bar{x}:\bar{l}_x::\bar{y}:\bar{l}]}$. By the induction hypothesis, the property of strong preservation holds for ψ and $\neg\psi$. By assumption, either $S(\psi)$ or $S(NF(\neg\psi))$ hold. If $S(\psi)$ holds, then ψ is true in any strong instance of $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. If $S(NF(\neg\psi))$ holds, then $\neg\psi$ is true in any strong instance of $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$, i.e., ψ is false in any such instance. By consequence, $(t_0$ *if* ψ *else* $0)$ has the same interpretation in all the strong instances of $L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$. For a given strong instance of $\bar{y} : \bar{l}$, there are $Mul(\bar{l}_x)$ such instances. Thus, $t^{I[\bar{y}:\bar{d}]} = (\sum_{\bar{x}\in\bar{T}}(Mul(\bar{l}_x) \times \chi(t_0)$ *if* $\psi))^{L[\bar{y}:\bar{l}]} = \chi(t)^{L[\bar{y}:\bar{l}]}$. Strong preservation follows from the fact that $\chi(t)$ is a number of multiplicity 1 in $L[\bar{y} : \bar{l}]$, and thus can be rotated by π any number of times. □

(weak) *If* $\forall\bar{x} \in \bar{T}^0 : s(t_0) \wedge Reg(\bar{y}) \wedge [S(\psi) \vee S(NF(\neg\psi))]$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any* i, $\sum_{\bar{x}\in\bar{T}:\psi} t_0^{I[\bar{y}:\pi^i(\bar{l})]} = \pi^i(\sum_{\bar{x}\in\bar{T}}(Mul(l_x) \times \chi(t_0)$ *if* $\psi)^{L[\bar{y}:\bar{l}]})$.

Proof. This property is a special case of the above, where $\bar{d}$ is $\pi^i(\bar{l})$. □

Special rules Recall that we use $\bar{\overline{t}}$ to denote the concatenation of tuples $\overline{t_i}$, i.e., to denote $(\overline{t_1}, \ldots, \overline{t_n})$.

Rule for $\exists\bar{x} \in \bar{T} : P(\bar{\overline{t}}, \bar{\overline{u}}) \wedge \psi$ (Rule 4.41 on page 81)

This rule improves compression of the domain by relaxing the constraint in Rule 4.35 that makes ϕ true in any strong instance of L'. It avoids the regularity constraints on the atoms in $P(\bar{\overline{t}}, \bar{\overline{u}})$).

(strong) *If* $(\exists\bar{x} \in \bar{T}^0 : S(\psi) \vee [Reg(\bar{y}) \wedge W(\psi)]) \vee (\exists\bar{x} \in \bar{T}^0 : Reg(\bar{\overline{u}}) \wedge W(P(\bar{\overline{t}}, \bar{\overline{u}}))) \wedge S(\psi))$ *is true in* $L' = L[\bar{y} : \bar{l}]$*, then,* $\exists\bar{x} \in \bar{T} : P(\bar{\overline{t}}, \bar{\overline{u}}) \wedge \psi$ *is true in every strong instance of* L'.

Proof. The proof for this rule has 2 parts. The first part assumes that the first disjunct is true; this disjunct is the translation of $S(\phi)$ of Rule (4.35) and this rule is already proven. The second part assumes that

$$\exists\bar{x} \in \bar{T}^0 : Reg(\bar{\overline{u}}) \wedge W(P(\bar{\overline{t}}, \bar{\overline{u}}))) \wedge S(\psi)$$

is true. By this hypothesis, $Reg(\bar{\overline{u}}) \wedge P(\chi(\bar{\overline{t}}, \bar{\overline{u}})) \wedge w(\bar{\overline{t}}) \wedge S(\psi)$ is true in $L' = L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$ for some l_x. Let's call $(\overline{\overline{l_t}, \overline{l_u}})$ the value of $\chi(\bar{\overline{t}}, \bar{\overline{u}})$ in that L'. Consider any strong instance $[\bar{y} : \bar{d}]$ of $[\bar{y} : \bar{l}]$. By assumption, $Reg(\bar{\overline{u}})$ is true in L': by Lemma 2 (noting that the variables of $\bar{\overline{u}}$ are in $\bar{y}$), there is a j such that

$\overline{\overline{u}}^{I[\bar{y}:\bar{d}]} = \pi^j(\overline{\overline{l_u}}) = \pi^j(\chi(\overline{\overline{u}})^{L'})$. Now consider $I' = I[\bar{x} : \pi^j(\bar{l}_x) :: \bar{y} : \bar{d}]$ for that j. Since the variables of $\bar{\bar{t}}$ are in $\bar{x}$, $\bar{\bar{t}}^{I'} = \bar{\bar{t}}^{I[\bar{x}:\pi^j(\bar{l}_x)]}$. A consequence of $W(P(\bar{\bar{t}}, \overline{\overline{u}}))$ is $w(\bar{\bar{t}})$. Hence, $\bar{\bar{t}}$ is weakly preserved, and $\bar{\bar{t}}$ evaluates to $\pi^j(\chi(\bar{\bar{t}})^{L'})$ in I'. By assumption, $P(\chi(\bar{\bar{t}}, \overline{\overline{u}}))$ is true in L', and thus $P^I(\pi^j(\chi(\bar{\bar{t}}, \overline{\overline{u}})^{L'})) = (P(\bar{\bar{t}}, \overline{\overline{u}}))^{I'}$ is true by construction of the p_i^I in P (Definition 25 on page 74 and Equation 4.20 on page 73). Similarly, the hypothesis that $S(\psi)$ is true in L' implies that ψ is true in I'. Hence, ϕ is true in $I[\bar{y} : \bar{d}]$. □

Rule for $\sum_{\bar{x}\in\bar{T}:p(\bar{t},\bar{u})\wedge\psi} t_0$ (Rule 4.42)

Given the proof for Rule 4.39, it suffices to show that

$$\frac{Mul(\bar{x}) \times Lcm(\chi(\bar{t}, \bar{u}))}{Lcm(\chi(\bar{t})) \times Lcm(\chi(\bar{u}))} \tag{4.47}$$

(evaluated in $L' = L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$) is the number X of strong instances $[\bar{x} : \bar{d}_x]$ of $[\bar{x} : \bar{l}_x]$ for which the filter formula is true in $I' = I[\bar{x} : \bar{d}_x :: \bar{y} : \bar{d}]$ when the translated filter formula is true in L', when $[\bar{y} : \bar{d}]$ is an instance of $[\bar{y} : \bar{l}]$, and when either:

(i, weak instance) $w(t)$ is true in L' and $\bar{d} = \pi^i(\bar{l})$,

(ii, strong instance) or $s(t)$ is true in L'.

Indeed, the value of t_0 and the truth value of ϕ (and thus the value of t_0 *if* ψ *else* 0) are strongly preserved (see proof of the rule for $\sum_{\bar{x}\in\bar{T}:\psi} t_0$ above).

Let's call $(\bar{l}_t, \bar{l}_u)$ the value of $\chi(\bar{t}, \bar{u})$ in L'. The weak expansion of this tuple has $Lcm(\bar{l}_t, \bar{l}_u)$ tuples, and those expanded tuples are in the expansion of p^L, i.e., they are the only ones to make p true in I'. Those tuples have 2 parts: one in the expansion of $\bar{l}_t$, the other in the expansion of $\bar{l}_u$.

Because of $Reg(\bar{t})$ and Lemma 2 on page 84, there are $Lcm(\chi(\bar{t}))^{L'}$ distinct tuples in the expansion of $\bar{l}_t$. There are $Mul(\bar{x})^{L'}$ different variable assignments to $\bar{x}$, so each value in the first part of the tuples in the expansion of $(\bar{l}_t, \bar{l}_u)$ is produced by

$$A = [\frac{Mul(\bar{x})}{Lcm(\chi(\bar{t}))}]^{L'}$$

different variable assignments to $\bar{x}$ (by the strong preservation of values, Proposition 11).

In the first case (i), $\bar{d}$ is one of the $\pi^i(\bar{l})$, and thus $\bar{d}_u$ is one of the $\pi^i(\bar{l}_u)$ by Lemma 1; in the second case (ii), $Reg(\bar{u})$ is true in L', and by Lemma 2, $\bar{d}_u$ is

also one of the $\pi^i(\bar{l}_u)$. In both cases, there are $Lcm(\bar{l}_u) = Lcm(\chi(\bar{u})^{L'})$ distinct values in the expansion of $\bar{l}_u$. Similar to the argument in the previous paragraph, each value in the second part of the tuples in the expansion of $(\bar{l}_t, \bar{l}_u)$ is produced by

$$B = [\frac{Mul(\bar{y})}{Lcm(\chi(\bar{u}))}]^{L'}$$

different variable assignments to $\bar{y}$.

We are interested in the value of the term t for one of the $C = Mul(\bar{l})$ strong instances $\bar{d}$ of $\bar{l}$.

Hence, the number X is

$$Lcm(\chi(\bar{t}, \bar{u})) \times A \times B \times \frac{1}{C} = Lcm(\chi(\bar{t}, \bar{u}))\frac{Mul(\bar{x})}{Lcm(\chi(\bar{t}))}\frac{Mul(\bar{y})}{Lcm(\chi(\bar{u}))}\frac{1}{Mul(\bar{y})}$$

evaluated in L'. This is a whole number because, in L', $Mul(\bar{x})$ is a multiple of $Lcm(\chi(\bar{t}))$ due to the regularity conditions on functions and $Lcm(\chi(\bar{t}, \bar{u}))$ is a multiple of $Lcm(\chi(\bar{u}))$.

The formula is illustrated in Example 37 on the next page.

Note: the rule cannot be generalized to terms of the form $\sum_{\bar{x}\in\bar{T}:P(\bar{\bar{t}},\bar{\bar{u}})\wedge\psi} t_0$. Consider $P(\bar{\bar{t}}, \bar{\bar{u}}) = p_1(x_1, y_1) \wedge p_2(x_2, y_2)$ that is true in L': there are $Lcm(x_1, y_1) \times Lcm(x_2, y_2)$ tuples that make it true in I' (and not $Lcm(x_1, y_1, x_2, y_2)$, as would be needed in the generalized proof).

Rule for $(\sum_{\bar{x}\in\bar{T}:p(\bar{t},\bar{u})\wedge\psi} t_0) \in \{0, n\}$ (Rule 4.43)

The proof is similar to the previous one. Here, we take advantage of the fact that the sum aggregate may be 0.

When $Reg(\bar{u})$ is not required, a variable assignment $\bar{d}$ to $\bar{y}$ generates a value for $\bar{u}$ in I' that is either equal to $\pi^i(\chi(\bar{l}_u))$ for some i, or different from it for any i. In the first case, the value of t_0 and ψ are strongly preserved by hypothesis, and the number of assignments that satisfy $p(\bar{t}, \bar{u})$ can be computed as in the proof of Rule (4.42): the sum aggregate is equal to n. In the second case, $p(\bar{t}, \bar{u})$ is false in any $I[\bar{x} : \bar{l}_x :: \bar{y} : \bar{d}]$ by definition of the expansion of the interpretation of predicates, and the sum aggregate is equal to 0.

Rule for $\sum_{\bar{x}\in\bar{T}:\bar{t}=\bar{u}\wedge\psi} t_0$ (Rule 4.44)

Recall that $\bar{t} = \bar{u}$ is a shorthand for $\bigwedge_i t_i = u_i$.

(strong) *If* $\forall \bar{x} \in \bar{T}^0 : (Reg(\bar{t}) \vee Reg(\bar{u})) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any strong instance* $[\bar{y} : \bar{d}]$ *of* $[\bar{y} : \bar{l}]$*, there exists an* i *such that* $\sum_{\bar{x} \in \bar{T} : \bar{t} = \bar{u} \wedge \psi} {t_0}^{I[\bar{y}:\bar{d}]} = \pi^i(\sum_{\bar{x} \in \bar{T}^0 : \chi(\bar{t}) = \chi(\bar{u}) \wedge S(\psi)} \frac{Mul(\bar{x})}{Lcm(\chi(\bar{u}))} \times \chi(t_0)^{L[\bar{y}:\bar{l}]})$.

The proof is similar to the previous ones. Here, we take advantage of the conjunction of equalities $\bigwedge_i \chi(\bar{t}_i) = \chi(\bar{u}_i)$ that is true in L' when the filter formula is true: then, $mul(\bar{t}_i) = mul(\bar{u}_i)$ and $Lcm(\chi(\bar{t})) = Lcm(\chi(\bar{u})) = Lcm(\chi(\bar{t}, \bar{u}))$ in L'.

The value of t_0 and the truth value of ϕ (and thus the value of t_0 *if* ψ *else* 0) are strongly preserved (see proof of Rule 4.39 for $\sum_{\bar{x} \in \bar{T} : \psi} t_0$ above).

Given the proof for Rule 4.39, it suffices to show that

$$\frac{Mul(\bar{x})}{Lcm(\chi(\bar{u}))} \tag{4.48}$$

(evaluated in $L' = L[\bar{x} : \bar{l}_x :: \bar{y} : \bar{l}]$) is the number X of strong instances $[\bar{x} : \bar{d}_x]$ of $[\bar{x} : \bar{l}_x]$ for which the filter formula is true in $I' = I[\bar{x} : \bar{d}_x :: \bar{y} : \bar{d}]$ when the translated filter formula is true in L', when $[\bar{y} : \bar{d}]$ is a strong instance of $[\bar{y} : \bar{l}]$, and when $(Reg(\bar{t}) \vee Reg(\bar{u})) \wedge s(t_0) \wedge [S(\psi) \vee S(NF(\neg\psi))]$ is true in L'.

Let's call $(\bar{l_t}, \bar{l_u})$ the value of $\chi(\bar{t}, \bar{u})$ in L'. By the induction hypothesis, Lemma 2 holds for the tuples $\bar{t}$ and $\bar{u}$. By assumption, the precondition $Reg(\bar{t})$ or $Reg(\bar{u})$ holds for one of the two tuples. By consequence, for any strong instance I' of L', there is an i such that $\bar{t}^{I'} = \pi^i(\bar{l_t}) \vee \bar{u}^{I'} = \pi^i(\bar{l_u})$. When the filter formula holds in L', $\bar{l_t} = \bar{l_u}$, and thus, $\bar{t}^{I'} = \pi^i(\bar{l_t}) = \pi^i(\bar{l_u}) = \bar{u}^{I'}$ for some i. Hence, the number of strong instances of $(\bar{l_t}, \bar{l_u})$ that satisfy the equality is $Lcm(\bar{l_t}) = Lcm(\bar{l_u}) = Lcm(\bar{l_t}, \bar{l_u})$ evaluated in L'.

By the same reasoning as in the proof of Rule 4.42, the number X is

$$\begin{aligned} X =& \frac{Mul(\bar{x}) \times Lcm(\chi(\bar{t}, \bar{u}))}{Lcm(\chi(\bar{t})) \times Lcm(\chi(\bar{u}))} \\ =& \frac{Mul(\bar{x})}{Lcm(\chi(\bar{t}))} = \frac{Mul(\bar{x})}{Lcm(\chi(\bar{u}))} \end{aligned}$$

Rule for $\sum_{\bar{x} \in \bar{T} : \bar{t} = \bar{u} \wedge \psi} t_0 \in \{0, n\}$ **(Rule 4.45)**

The proof can be obtained by taking advantage of the fact that the sum aggregate may be 0, and of the equality $\bar{t} = \bar{u}$, as done in the previous proofs.

This completes the proof by structural induction.

Example 37. *Consider the following lifted vocabulary with singleton types:*

$$type\ X \triangleq \{lx\}$$

$$type\ Y \triangleq \{ly\}$$

$$type\ T_1 \triangleq \{lt_1\}$$

$$type\ T_2 \triangleq \{lt_2\}$$

$$type\ U \triangleq \{lu\}$$

$$t_1 : X \to T_1$$

$$t_2 : X \to T_2$$

$$u : Y \to U$$

$$p : T_1 \times T_2 \times U \to Bool$$

and the atom $p(t_1(x), t_2(x), u(y))$ *where* x, y *are variables of types* X, Y*. Its translation is also* $p(t_1(x), t_2(x), u(y))$*.*

Consider the regular lifted structure L *with the following interpretations:*

$$t1 \triangleq \{lx \to lt_1\}$$

$$t2 \triangleq \{lx \to lt_2\}$$

$$s \triangleq \{ly \to lu\}$$

$$p \triangleq \{lt_1, lt_2, lu\}$$

$$mul \triangleq \{lx \to 6, ly \to 4, lt_1 \to 2, lt_2 \to 3, lu \to 2\}$$

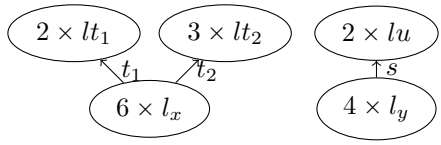


Figure 4.4: Simple model illustrating the special rule for sum aggregates

Reg$(t_1(lx), t_2(lx))$ *is true because* $mul(lx) = lcm(lx)$*. Similarly, Reg*$(u(y))$ *is true.*

The expansion of (lt_1, lt_2, lu) *(which is true in the lifted interpretation of* p*) has* $Lcm(lt_1, lt_2, lu) = lcm(2, 3, 2) = 6$ *tuples:*

$$(dt_1^1, dt_2^1, du^1), (dt_1^2, dt_2^2, du^2)$$

$$(dt_1^1, dt_2^3, du^1), (dt_1^2, dt_2^1, du^2)$$

$$(dt_1^1, dt_2^2, du^1), (dt_1^2, dt_2^3, du^2)$$

There are $mul(lx) = 6$ *concrete instances of* $[x : lx]$*, and there are* $Lcm(lt_1, lt_2) = lcm(2, 3) = 6$ *tuples in the weak expansion of* $(t_1(lx), t_2(lx))$*:*

$$(t_1(dx^1), t_2(dx^1)) = (dt_1^1, dt_2^1)$$

$$(t_1(dx^2), t_2(dx^2)) = (dt_1^2, dt_2^2)$$

$$(t_1(dx^3), t_2(dx^3)) = (dt_1^1, dt_2^3)$$

$$(t_1(dx^4), t_2(dx^4)) = (dt_1^2, dt_2^1)$$

$$(t_1(dx^5), t_2(dx^5)) = (dt_1^1, dt_2^2)$$

$$(t_1(dx^6), t_2(dx^6)) = (dt_1^2, dt_2^3)$$

Each value (dt_1^i, dt_2^j) *is produced by* $A = Mul(\bar{x}) \div Lcm(\bar{t}) = mul(x) \div Lcm(t_1, t_2) = 6/6 = 1$ *different variable assignments for* x*.*

Similarly, there are $mul(ly) = 4$ *concrete instances of* $[y : ly]$*, and there are* $Lcm(lu) = lcm(2) = 2$ *tuples in the weak expansion of* $(u(y))$*. Each value* du^i *is produced by* $B = Mul(\bar{y}) \div Lcm(\bar{u}) = mul(y) \div Lcm(s) = 4/2 = 2$ *different variable assignments for* y*. There are* $C = mul(ly) = 4$ *such variables assignments.*

The number of instances of $[x : lx]$ *for which* $p(t_1(x), t_2(x), u(y))$ *is true in* $I[x : dx :: y : dy]$ *when* $p(t_1(x), t_2(x), u(y))$ *is true in* $L[x : lx :: y : ly]$ *and* $[y : dy]$ *is an instance of* $[y : ly]$ *is:*

$$Lcm(t_1, t_2, s) \times A \times B \times \frac{1}{C} = 6 \times 1 \times 2 \times \frac{1}{4} = 3$$

For the instance $[y : dy^3]$*, they are* $[x : dx^1], [x : dx^3], [x : dx^5]$*.*

4.4 Experimental evaluation

The goal of the evaluation is to show that there are satisfiability problems where substantial compression of the domain is possible, and that the lifted models can indeed be expanded into concrete ones.

Problems are expressed in FO(·). A FO(·) (lifted) sentence is translated by IDP-Z3 (Carbonnelle et al., 2022) for use with the Z3 SMT solver (De Moura & Bjørner, 2008). The overhead of this translation is negligible.

The source code used for our evaluation is available on GitLab to allow reproduction.[7] Tests were run using Z3 v4.12.1 on an Intel Core i7-8850H CPU at 2.60GHz with 12 cores, running Ubuntu 22.04 with 16 GB RAM. We run an extended version of IDP-Z3 v0.10.11 on Python 3.11, that supports model expansion of FO(·) theories in which some types are not interpreted, as explained in the next paragraph and, in more details, in Section 4.4.2.

Because the size of the lifted domain is not known, lifted models are found iteratively: starting with an empty lifted domain, the lifted sentence is reduced to a propositional sentence and its satisfiability is determined with a standard satisfiability solver capable of arithmetic reasoning. If the sentence is unsatisfiable with this domain, the sentence is reduced to a minimal unsatisfiable formula (Lynce & Marques-Silva, 2004), and the domains of the types used in that formula are extended with one element (unless the lifted type already has the given size of the concrete type), so that another reduction to a propositional sentence can be done. This process is repeated until a model of the sentence is found, or until all the lifted types have the given size of the concrete types (in which case the sentence is unsatisfiable for those sizes). This iterative process is more efficient than the non-iterative process obtained by assuming a large enough lifted domain.

In many experiments, it was sufficient to support the special Rule (4.42) for the case where the atom p is of the form $t_1 = s$ (e.g., $holeOf(p) = h$). In that case, $mul(\chi(t_1)) = mul(\chi(s)) = Lcm(\chi(t_1, s))$ when the filter of the sum aggregate is true, and the transformation of the sum aggregate per Rule (4.42) simplifies to:

$$\frac{1}{mul(\chi(s))} \sum_{\bar{x}\in\bar{T}} (Mul(\bar{x}) \times \chi(t) \ \textit{if}\ 0 < Mul(\bar{x}) \wedge \chi(t_1){=}\chi(s) \wedge \chi(\phi)) \qquad (4.49)$$

This formula does not use the *lcm* function, which is not supported natively by solvers. In other experiments, we used an interpretation table for *lcm*.

[7] `https://gitlab.com/pierre.carbonnelle/idp-z3-generative`

4.4.1 Pigeonhole problems

To validate the approach, we first consider the satisfiability problem of assigning each pigeon to one pigeonhole, such that each pigeonhole holds exactly 2 pigeons. A function $holeOf : Pigeon \mapsto Hole$ is used to represent this relation. We only consider the satisfiable case, i.e., those in which there are twice as many pigeons as holes.

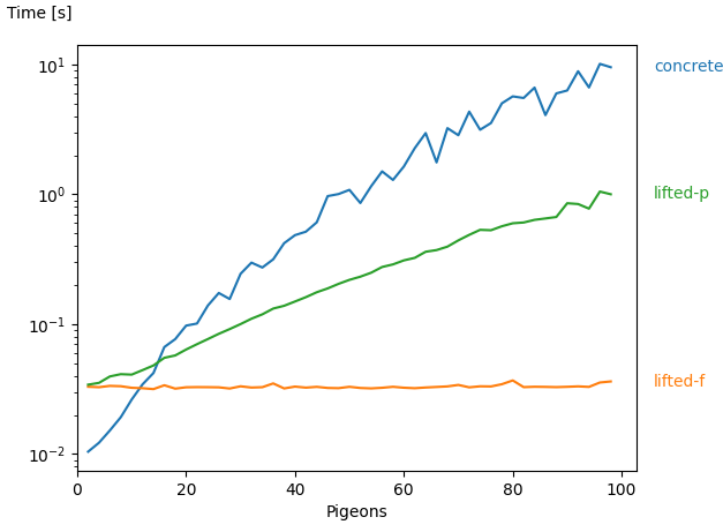


Figure 4.5: Wall clock time to solve our pigeonhole problem

The results are shown in Figure 4.5. With the original sentence, the solution time increases exponentially ("concrete" line in Figure 4.5). We are aware that symmetry breaking can reduce the complexity, but to the best of our knowledge, solving time is at least linear in the number of pigeons for the best symmetry breaking solvers that are available.

The lifted solution has only 1 lifted pigeon and 1 lifted hole, as shown in the introduction of the chapter. As expected, the time needed to solve the lifted problem is essentially constant when it is satisfiable ("lifted-f" line in Figure 4.5). The correct multiplicities are found quickly by Z3 using a sub-solver specialized for arithmetic. For example, with 10,000 pigeons, the lifted sentence is solved in only 0.05 sec and the expansion of the lifted model into a concrete one in 0.1 sec.

For comparison, using the constraint satisfaction solver Gecode 6.3.0,[8], the concrete problem is solved in 0.53 sec for 50 pigeons; in 0.91 sec for 100 pigeons; in 8 sec for 200 pigeons; and in 18 sec for 300 pigeons. Similar results are obtained with the kodkod solver (Torlak & Jackson, 2007).

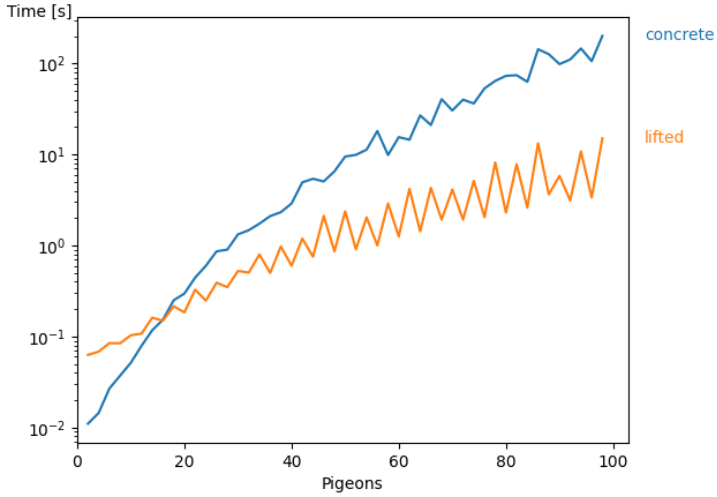


Figure 4.6: Wall clock time to solve our "2-2 pigeonhole" problem

We also expressed the same problem using a predicate $isIn : Pigeon \times Hole \to \mathbb{B}$ instead of the function $holeOf$, and an additional constraint to ensure that a pigeon is assigned to only one pigeonhole. The sentences are transformed as in Equation 4.46 on page 82 (Example 36). There are no regularity constraints for functions anymore (Equations 4.24-4.25). The transformed aggregates now contain the *Lcm* function, which is implemented using an interpretation table. Experiments confirm compression of the domain and equisatisfiability, but the complexity of solving the lifted problem is at least quadratic with the number of pigeons (line "lifted-p" in Figure 4.5 on the facing page) because the *lcm* interpretation table grows quadratically with it. Still, this lifted problem is solved faster than the concrete problem.

We also validated our method on pigeonhole problems where the relation between pigeon and holes is a bit more complex: a pigeon is hosted in exactly 2 holes, which can hold exactly 2 pigeons. The number of pigeons is even. Again, the relation is represented by binary predicate *isIn* and the transformed sentences

[8] https://www.gecode.org/

contain the *Lcm* function. The lifted solution now has 2 lifted pigeons and 2 lifted holes. More iterations are needed to find the lifted solution than in the simpler problem where the lifted solution has only 1 lifted pigeon and 1 lifted hole. Experiments confirm equisatisfiability, but the complexity of solving the lifted problem is at least quadratic with the number of pigeons (Figure 4.6).

Finding a more efficient implementation for *Lcm* has been left for future work. One option is to use so-called "user propagators", i.e., custom theory solvers outside Z3 (Bjørner et al., 2021).

Finally, we have conducted experiments on an optimization problem: assign each pigeon to one pigeonhole so that each hole has at most 2 pigeons, with a minimal number of pigeonholes. Only the number of pigeons is given, and it may be odd. This is a simple example of a *generative configuration problem*, discussed in more details in the next section. In the concrete case, the number of pigeonholes is found by an iterative process: this process starts by assuming that the number of pigeonholes is one, and it increases this number until a solution is found. The number of pigeonholes is then minimal. To solve the lifted problem, a similar iterative approach is used, resulting in a solution with a minimal number of lifted pigeonholes. Solving time for increasing numbers of pigeons is shown in Figure 4.7 on the next page, for both the concrete and lifted problem. Solving time of the concrete problem grows quickly because of the iterative process needed to find the number of pigeonholes: more iterations are needed as the number of pigeons grows, and the search becomes more complex at each iteration. (Performance could be improved by using better solvers)

The lifted solution found by the iterative process for 5 pigeons is shown in Figure 4.8a on the facing page. Note that the number of *concrete* pigeonholes is not minimal: a solution exists with 3 pigeonholes (Figure 4.8b). The number of concrete pigeonholes in the lifted solution found by the iterative process (here: 5) is an upper bound to the number of concrete pigeonholes in an optimal solution (here: 3). The number of concrete pigeons found by the iterative process for the lifted problem can be improved by performing additional iterations (to increase the number of lifted pigeons and pigeonholes), where each iteration searches for a model minimizing the number of concrete pigeonholes given the number of lifted pigeons and pigeonholes. In this case, the minimum number of concrete pigeonholes is found in a couple of additional iterations: the optimal solution (Figure 4.8b) is still compressed. But, in general, there is no guarantee that additional iterations would not result in a lower number of concrete pigeonholes, unless the lifted domain has no compression anymore (all multiplicities in it are 1).

If a suboptimal solution is not acceptable, it may be beneficial to use a 2-stage approach: (i) use lifted search to find an upper bound to the minimal number of

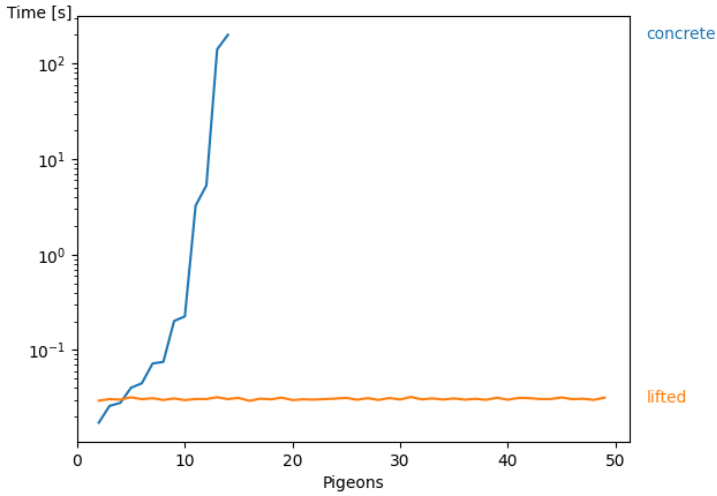
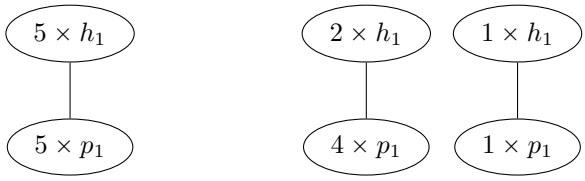


Figure 4.7: Wall clock time to solve a pigeonhole optimization problem

pigeonholes (in our example: 5); (ii) perform model optimization in the concrete domain, using the upper bound on the number of pigeonholes found in step (i).



(a) with 5 pigeonholes (b) with 3 pigeonholes

Figure 4.8: Lifted solutions for a pigeonhole optimization problem

4.4.2 Generative configuration problems

A generative configuration problems (GCP) is a configuration problem in which the number of some components is not known and has to be found. In several papers (Reger et al., 2016; Aschinger et al., 2014; Comploi-Taupe et al., 2022; Schenner & Taupe, 2017; Taupe et al., 2016; Friedrich et al., 2011), such a problem is solved by making an assumption on the number of components, and by reducing the sentence to be satisfied to a propositional formula. If

the assumed number of components is too low, the propositional formula is unsatisfiable, and the assumed number is revised upwards until the formula becomes satisfiable. Some variations of this basic iterative method have been developed, e.g., for multi-sorted logic (Reger et al., 2016).

Unfortunately, these methods do not scale well (Schenner & Taupe, 2017) because the search space grows exponentially with the domain size. If the initial domain size is too high, the search space is much larger than needed. If the initial domain size is too low, many iterations are needed. Both have a negative impact on performance. For best performance, the initial domain size should be as close as possible to the number of components in a solution. Thus, to improve performance, attempts have been made to estimate the lower and upper bounds of cardinalities of components from the description of the problem in a fragment of first-order logic (Aschinger et al., 2014) or in a class diagram (Taupe et al., 2016).

We propose instead to obtain the number of components by solving the lifted problem. When the compressed domain is smaller than the concrete domain, the number of iterations needed to solve the lifted sentence is smaller than the number of iterations needed for the original sentence, leading to better performance.

We evaluate our solving method on three representative GCP discussed in the literature:

- the House Configuration (HCP) and Reconfiguration (HRP) problem (Friedrich et al., 2011),
- the Organized Monkey Village (Reger et al., 2016)
- the Rack problem (Feinerer et al., 2011; Comploi-Taupe et al., 2022), already introduced in Section 1.1.3 on page 3.

These problems are represented by a theory (i.e., a set of sentences) in a vocabulary that uses a different type for each type of components in the configuration. As an example, the knowledge base for the Rack problem (Listing 4.1) uses a vocabulary with the *Rack*, *Frame*, *Module* and *Element* types.

Listing 4.1: Knowledge base for the Rack problem

```
vocabulary V {
    type Rack
    type Frame
    type Module
```

```
    type Element

    type RackType ≜ { RackSingle, RackDouble }
    type ModuleType ≜ { ModuleI, ModuleII, ModuleIII,
                        ModuleIV, ModuleV }
    type ElementType ≜ { ElementA, ElementB, ElementC,
                         ElementD }

    module14: Module → 𝔹

    typeOfR: Rack → RackType
    typeOfM: Module → ModuleType
    typeOfE: Element → ElementType

    rackOfF: Frame → Rack
    frameOfM: Module → Frame
    elementOfM: module14 → Element
}

theory T:V {
    // definition of the domain of elementOfM
    { ∀ m ∈ Module: module14(m) ← typeOfM(m) ≠ ModuleV.}

    // A rack has 4 or 8 frames
    ∀ r ∈ Rack: typeOfR(r) = RackSingle ⇒
                  #{f ∈ Frame: rackOfF(f)=r} = 4.
    ∀ r ∈ Rack: typeOfR(r) = RackDouble ⇒
                  #{f ∈ Frame: rackOfF(f)=r} = 8.

    // A frame can have up to 5 modules
    ∀ f ∈ Frame: #{m ∈ Module: frameOfM(m)=f} ⩽ 5.

    // A Frame containing a ModuleII must have also exactly
    one ModuleV
    ∀ f ∈ Frame:
      (∃ m2 ∈ Module: frameOfM(m2)=f ∧ typeOfM(m2)=ModuleII)
       ⇔ #{m5 ∈ Module:frameOfM(m5)=f∧typeOfM(m5)=ModuleV}=1.

    // A Frame containing a ModuleV must have at least a
    ModuleII
    ∀ f ∈ Frame:
      ¬(∃m2 ∈ Module: frameOfM(m2)=f∧typeOfM(m2)=ModuleII)
      ⇒ #{m5 ∈ Module:frameOfM(m5)=f∧typeOfM(m5)=ModuleV}=0.

    // #modules required per element
    ∀ e ∈ Element: typeOfE(e)=ElementA
```

```
      ⇒ #{m ∈ Module:elementOfM(m)=e∧typeOfM(m)=ModuleI}=1.
    ∀ e ∈ Element: typeOfE(e)=ElementB
      ⇒ #{m ∈ Module elementOfM(m)=e∧typeOfM(m)=ModuleII}=2.
    ∀ e ∈ Element: typeOfE(e)=ElementC
      ⇒#{m ∈ Module:elementOfM(m)=e∧typeOfM(m)=ModuleIII}=3.
    ∀ e ∈ Element: typeOfE(e)=ElementD
      ⇒ #{m ∈ Module:elementOfM(m)=e∧typeOfM(m)=ModuleIV}=4.
}

theory S:V { // input structure for A1
    #{e ∈ Element: typeOfE(e) = ElementA} = 1.
    #{e ∈ Element: typeOfE(e) = ElementB} = 0.
    #{e ∈ Element: typeOfE(e) = ElementC} = 0.
    #{e ∈ Element: typeOfE(e) = ElementD} = 0.
}
```

A model of this theory is found using two iterative methods:

- ("t_1") The upper bound of the size of each type of components is initially assumed to be 1; this number is incremented for the types used in the unsatisfiable core of the theory, until a model is found.
- ("t_2") Similar to "t_1", but the assumed number is multiplied by 1.5 (instead of incremented) at each iteration.

Note that these iterative methods may result in a type having more elements than needed for a solution. For example, a model found with t_1 may have 9 elements of type *Rack*, even though the configuration requires only 7; two are "unused". Thus, it is necessary to introduce a predicate for each type to indicate which elements of the type represent actual components of the configuration, as illustrated for the *Rack* type below.

```
type Rack_           // racks, including unused
Rack: Rack → 𝔹       // racks used in a configuration
```

The quantifications and aggregates in the theory (e.g., ∀ `r` ∈ `Rack:...`) consider the used elements only.

To allow the use of other iterative methods that do not need to compute the unsatisfiable core of the theory, we also translate this theory to use another vocabulary. This translation is done automatically. In this new vocabulary, the components of the solutions are now elements of a generic type, called "ConfigObject". Predicates define subtypes of this type. Part of the new vocabulary for the Rack problem is thus as follows:

```
type ConfigObject
Rack, Frame, Module, Element: ConfigObject → 𝔹
```

The theory in this new vocabulary is solved using these two iterative methods:

- ("m_1") The exact number of elements in *ConfigObject* is initially assumed to be 1; this number is incremented until a model is found. Each element of *ConfigObject* represents a component of the configuration.
- ("m_2") The upper bound of the number of elements in *ConfigObject* is initially assumed to be 1; this number is multiplied by 1.5 until a model is found; the interpretation of a predicate determines which elements in *ConfigObject* represent actual component(s) of the configuration.

To evaluate our method described in the previous section, we automatically translate these theories into a lifted version. Only the component types are lifted: the identifiers of the other types are constant (e.g., `RackSingle` in `RackType`) and thus have a multiplicity of one. Notice that only the equality predicate and unary symbols are used in the theory: hence, most regularity constraints introduced by our method are trivially satisfied. We then apply the iterative methods to solve them. The names of the lifted methods are prefixed by "l" (thus, lt_1, lt_2, lm_1, lm_2). The number of lifted elements is found iteratively, and their multiplicities are searched by the solver in each iteration, until the lifted sentence is satisfiable. Finally, we verify that an expansion of the models of the lifted sentence satisfies the concrete sentence (this step is not included in time measurements).

A lifted solution for the Rack ABCD 12 problem is illustrated in Figure 4.9. Each node in the diagram represents a lifted component and its multiplicity. The arrows represent the unary functions linking the components (they are not labeled due to space constraints). The elements of `RackType, ModuleType` and `ElementType` are not shown.

It is not practical to show the full expansion of this model. The expansion of the left branch only is shown in Figure 4.10.

For comparison, we also used another constraint programming (CP) language (CPMpy Guns, 2019) and another CP solver (OR-tools[9]) for some of our experiments. However, we did not develop the automatic transformation above for them, and, as a consequence, our results with them are patchy (but representative).

[9]https://google.github.io/or-tools/

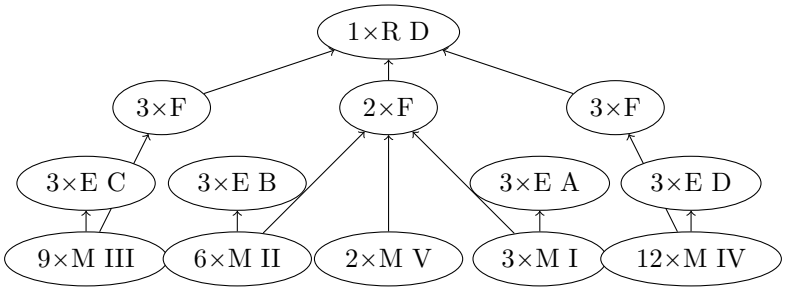


Figure 4.9: Lifted configuration for the Rack ABCD 12 problem. 1 Rack contains 8 Frames, 3 of which contain 9 Modules of type III, for 3 Elements of type C.

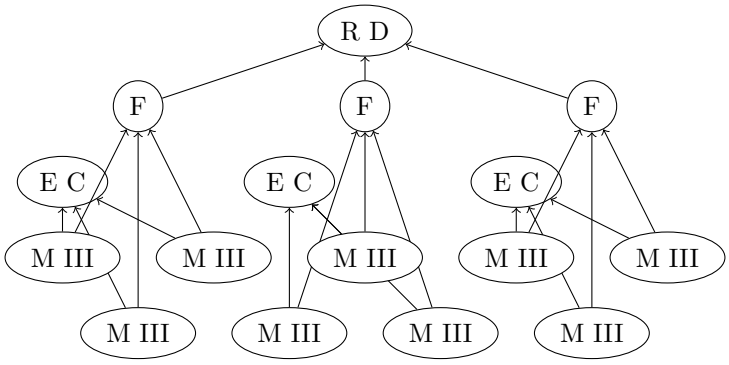


Figure 4.10: Partial expansion of the Rack ABCD 12 solution

The top half of Table 4.4 on the next page shows results with Z3 (with automatic translation of the sentence), the bottom half with the OR-tools solver. T denotes a timeout after 200 seconds. Performance varies due, e.g., to non-determinism of the unsat-core algorithm in Z3; results from a representative run are shown. The best results in the run are shown in bold.

Solutions to problems with higher suffixes have more components than similar problems with lower suffixes. The table shows near constant time performance on the Rack ABCD problems. The lifted methods solve each of the 20 occurrences of the ABCD Racks problem in Comploi-Taupe et al. (2022) in less than 5 seconds (instead of 6 minutes on average in that paper, using an ASP solver).

These results show that our method has significant merits for solving certain classes of model expansion problems.

	Performance								#	
Lifted?	Yes (our approach)				No				Yes	No
Method name	lm_1	lm_2	lt_1	lt_2	m_1	m_2	t_1	t_2		
Typed?	no	no	yes	yes	no	no	yes	yes		
Domain change	+1	×1.5	+1	×1.5	+1	×1.5	+1	×1.5		
Problem										
HCP 1	0.34	0.25	0.11	**0.1**	0.64	0.53	0.20	0.15	6	10
HCP 3	0.31	0.26	**0.11**	**0.11**	T	1.98	T	T	6	27
HRP 1	0.59	1.00	0.15	**0.13**	1.00	0.85	0.24	0.20	7	10
HRP 3	0.71	0.63	**0.14**	0.15	161.12	4.55	T	T	8	25
Monkey 1	0.30	0.24	**0.11**	**0.11**	T	195.73	T	T	6	20
Monkey 4	0.33	0.31	**0.11**	**0.11**	T	T	T	T	6	52
Rack 2011	0.40	0.42	**0.06**	**0.06**	11.64	9.85	T	12.60	5	23
Rack A5	0.73	1.00	**0.21**	**0.21**	6.38	4.01	1.11	2.20	5	19
Rack A10	0.63	0.8	**0.27**	**0.27**	97.44	38.91	T	T	5	38
Rack A1000	0.37	0.36	**0.07**	0.08	T	T	T	T	4	2250
Rack ABCD4	29.89	5.40	**1.49**	2.59	123.16	81.95	28.45	15.33	13	20
Rack ABCD8	185.65	7.57	**1.69**	4.61	T	T	T	T	13	40
Rack ABCD20	T	47.18	**2.21**	9.53	T	T	T	T	13	100
Rack A1	**0.61**				0.75				5	7
Rack A2	**0.64**				1.54				5	9
Rack A3	**0.62**				3.69				5	11
Rack A4	**0.65**				36.9				5	13
Rack A5	**0.65**				T				5	19

T: timeout after 200 sec.

Table 4.4: Wall clock time in seconds to solve configuration problems, and number (#) of used components in models.

The iterative methods often introduce more lifted domain elements than needed for a solution, in which case unneeded domain elements have multiplicity 0. The unneeded elements introduce symmetries in the solution. We have investigated whether this symmetry can be reduced by symmetry breaking rules. We have considered two different symmetry breaking rules:

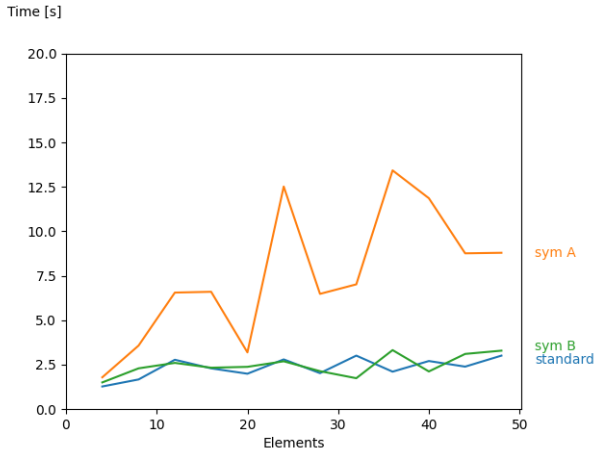
$$mul(l_0) \leqslant mul(l_1) \leqslant \cdots \leqslant mul(l_n) \tag{4.50}$$

$$(used(l_0) \Rightarrow used(l_1)) \wedge \cdots \wedge (used(l_{n-1}) \Rightarrow used(l_n)) \tag{4.51}$$

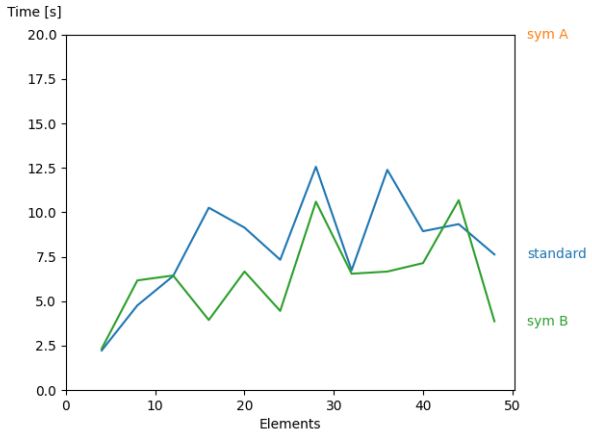
where $l_0, l_1, \ldots, l_n$ are the elements of a type.

Results are shown in Figure 4.11 on the following page for lt_1 and lt_2 ("sym A" stands for Equation 4.50, "sym B" for 4.51). We see that the first symmetry breaking rule has an adverse effect, while the second one has a positive effect on lt_2.

Symmetries remaining in the lifted domain cannot be eliminated by applying the lifting method a second time. Indeed, the lifted theory is designed such that the domain of each lifted model is a backbone of an automorphism of a concrete



(a) lt_1



(b) lt_2

Figure 4.11: Time to solve Rack ABCD with symmetry breaking

model. Furthermore, the iterative process finds a minimal lifted domain. Hence, the domain cannot be further compressed into a smaller backbone.

4.4.3 Social Golfer problem

Due to their many symmetries, Social Golfer problems are commonly used as benchmarks for symmetry breaking techniques (Liu et al., 2019). A Social Golfer problem is characterized by three numbers, g, s and w. In a problem, $N = g * s$ golfers play golf in g groups of s golfers once a week, each golfer playing in exactly one group every week. The task is to assign the golfers to groups for w weeks without any two golfers playing together in a group more than once.

Example 38. *A solution to the 4-4-2 Social Golfer problem is (where each number identifies a golfer):*

- *Week 1: (1,2,3,4)(5,6,7,8)(9,10,11,12)(13,14,15,16);*
- *Week 2: (1,6,9,15)(2,5,10,16)(3,7,12,14)(4,8,11,13).*

The time needed to solve a natural formulation of the problem in FO[·] for a given number of weeks increases when the number of golfers increases. We show that the lifting method combined with other symmetry breaking techniques significantly improves the performance. This illustrates the benefits of problem reformulation. Other reformulations of the Social Golfer problem have been explored by Smith (2001).

Listing 4.2 shows a natural representation of the 4-4-2 problems in FO[·]. The vocabulary has these function symbols:

- function `f` (resp. `s, t, ff`) maps a foursome to the first (resp. second, third, fourth) golfer in it;
- function `week` maps a foursome to a week.

To break some symmetries, we use types having an order, and we introduce a symmetry breaking constraint stating that the first, second, third and fourth golfers in a foursome are in strictly increasing order; the constraint has the consequences that the four golfers in a foursome are different, which is a requirement. Each golfer plays once a week. The number of times a golfer plays with another is zero or one. We finally add symmetry breaking rules stating that the foursomes of the first week simply are (1,2,3,4), (5,6,7,8), ..., (Liu et al., 2019) and that the remaining foursomes are in lexicographic order of the week, first, second, third and fourth golfer.

Listing 4.2: 4-4-2 Social Golfer problem

```
vocabulary V{
  type Golfer
  type Week
  type Foursome

  f,s,t,ff: Foursome → Golfer // 1st, 2nd, 3rd, 4th
  week : Foursome → Week

  var g in Golfer
  var fs in Foursome
  var w in Week
}
theory T:V {

  // The golfers in a foursome are different.
  ∀ fs: f(fs) < s(fs) < t(fs) < ff(fs).

  // Each golfer plays once a week
  ∀ g, w: #{fs : week(fs) = w ∧ f(fs)=g}
        + #{fs : week(fs) = w ∧ s(fs)=g}
        + #{fs : week(fs) = w ∧ t(fs)=g}
        + #{fs : week(fs) = w ∧ ff(fs)=g}
        = 1.

  // a golfer does not play more than once with another.
  ∀ g1, g2: #{fs :
      (g1=f(fs) ∧ g2 ∈ {s(fs), t(fs), ff(fs)})
    ∧ (g1=s(fs) ∧ g2 ∈ {f(fs), t(fs), ff(fs)})
    ∧ (g1=t(fs) ∧ g2 ∈ {f(fs), s(fs), ff(fs)})
    ∧ (g1=ff(fs) ∧ g2 ∈ {f(fs), s(fs), t(fs)}) } ⩽ 1.

  // first foursomes are (1,2,3,4)(5,6,7,8)...
  ∀ fs in Foursome: fs ⩽ 16/4 ⇒ week(fs)=1
      ∧ f(fs)=(fs-1) × 4+1 ∧  s(fs)=(fs-1) × 4+2
      ∧ t(fs)=(fs-1) × 4+3 ∧ ff(fs)=(fs-1) × 4+4.

  // foursome in lexicographic order
  ∀ fs1, fs2: 16/4 < fs1 < fs2 ⇒
    week(fs1)<week(fs2)
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)<f(fs2))
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)=f(fs2)∧s(fs1)<s(fs2))
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)=f(fs2)∧s(fs1)=s(fs2)
        ∧t(fs1)<t(fs2))
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)=f(fs2)∧s(fs1)=s(fs2)
        ∧t(fs1)=t(fs2)∧ff(fs1)<ff(fs2)).
```

```
    // sizes of concrete types.
    Week ≜ {1..2}                    // w
    #{g | g ∈ Golfer} = 16     // g*s
    #{fs | fs ∈ Foursome} = 8 // g*w
}
```

A model of this theory adapted for the simple 1-4-1 problem is illustrated in Figure 4.12.

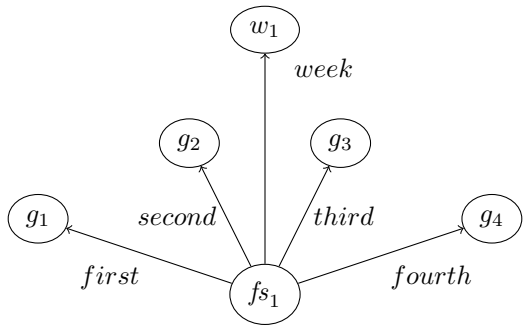


Figure 4.12: Solution for the 1-4-1 Social Golfer problem

The lifted vocabulary also uses numbers to identify golfers, weeks and foursomes, so that symmetry breaking constraints can be written[10]. Note, however, that these numbers are actually identifiers of domain elements, and their multiplicities do not have to be one. Arithmetic operations on these identifiers do not make sense.

Let's translate the statement that each golfer plays once a week. The filter in the first sum aggregate, i.e., in `#{fs : week(fs) = w ∧ f(fs)=g}`, is a tuple equality `(week(fs), f(fs)) = (w, g)` which satisfies the condition to apply the special rule (4.44) because $Reg(\bar{t}) = Reg(week(fs), f(fs))$ is satisfied (because $Lcm(fs) = mul(fs)$).

To avoid the need to compute *lcm* in the compression factor of the sum aggregate, we impose that the multiplicity of each week is one, and that the multiplicity of each golfer in a foursome is the multiplicity of the foursome, i.e.,

```
∀ w: mulW(w) = 1.
∀fs: mulFS(fs) = mulG(f(fs)) = mulG(s(fs))
                = mulG(t(fs)) = mulG(ff(fs)).
```

[10]Non-numeric types have no order in FO[·]

As a consequence of these constraints, the decompression factor for the first sum aggregate,

```
#{fs : week(fs) = w ∧ f(fs)=g}
```

is, per rule (4.42 on page 81) for $\bar{t} = (week(fs), f(fs))$ and $\bar{u} = (w, g)$:

$$\begin{aligned}
&\frac{Lcm(week(fs), f(fs), w, g) \times mul(fs)}{Lcm(week(fs), f(fs)) \times Lcm(w, g)} \\
=&\frac{Lcm(f(fs), g) \times mul(fs)}{Lcm(f(fs)) \times Lcm(g)} \\
=&\frac{mul(fs) \times mul(fs)}{mul(fs) \times mul(fs)} \\
=&1
\end{aligned}$$

Under these assumptions, the transformation of the first sum aggregate is then:

```
sum{{ 1 | fs in Foursome: week(fs) = w ∧ f(fs)=g}}
```

In other words, it is essentially unchanged. The condition $s(t)$ in Rule (4.42) reduces to **true** because ψ is formula **true**, because $Reg(\bar{t})$ is satisfied ($\bar{t}$ has only one variable, and $mul(fs) = lcm(fs)$), and because $Reg(\bar{u})$ is true (because the multiplicity of a week is 1).

The translation of the other aggregates also benefits from our assumptions. Note also that our assumptions entail that functions `f, s, t, ff` are regular, e.g.,

```
∀ fs: ∃ n in Int: mul(fs) = n  ×  mul(f(fs)).
```

The translation of the theory is listed in Appendix C on page 133. A model of the lifted theory for the 8-4-1 problem is shown in Figure 4.13. It is strikingly similar to the model of the concrete theory shown in Figure 4.12.

The iterative method we use to solve the lifted theory adds four golfers and one foursome at each iteration. As explained in the previous section, it can be beneficial to add symmetry breaking rules for the unneeded elements introduced by the iterative method. The best performance is obtained by using the symmetry breaking rule of Equation 4.50 on page 105.

The solving performance using IDP-Z3 can be summarized as follows:

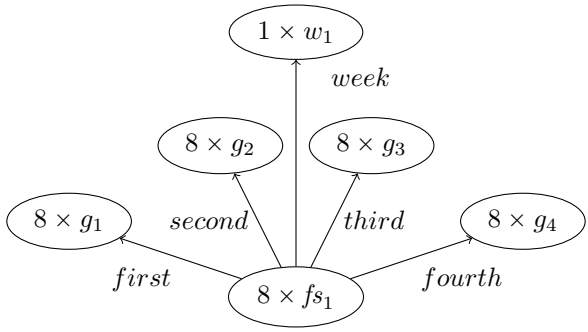
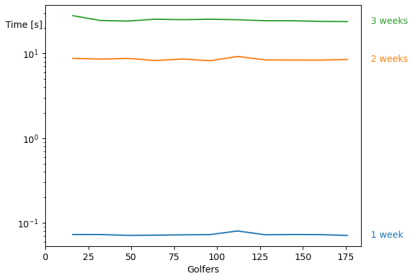
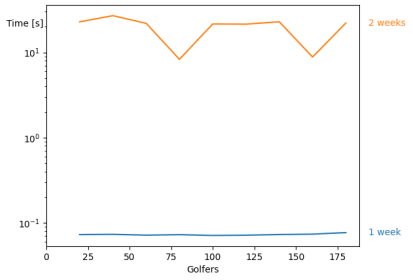


Figure 4.13: Lifted solution for the 8-4-1 Social Golfer problem

- the time to solve the trivial n-4-1 problems of scheduling only 1 week is essentially independent of the number of golfers (unlike the time to solve the original theory): it is about 0.1 second;
- the time to solve the 4n-4-2 and 4n-4-3 problems (with 16n golfers) is essentially independent of the number of golfers when this number is a multiple of 16: it is around 10 seconds for 2 weeks, and 25 seconds for 3 weeks (Figure 4.14a);
- similarly, the time to solve the 5n-4-2 problems (with 20n golfers) is essentially independent of the number of golfers, and is around 20 seconds (Figure 4.14b);



(a) 16n golfers



(b) 20n golfers

Figure 4.14: Wall clock time to solve some Social Player problems

While other solvers are faster at solving the 16-4-2, 16-4-3 and 20-4-2 problems, none have the constant complexity for multiple of 16 and 20 players that our solver has.

We already observed that concrete and lifted models are strikingly similar (Figures 4.12 and 4.13). Notice also that all the statements of the original theory are also in the lifted theory, and that the *mul* function only appears in the last two sentences of the lifted theory (the two sentences about the number of golfers and foursomes). Hence, to obtain better performance, one could use a 2-stage approach:

- solve the lifted theory for an increasing number of golfers and foursomes, assuming all multiplicities are one, i.e., solve the concrete theory to obtain minimal template models (as illustrated in Figure 4.12), using best-in-class SGP solvers;
- then, find models of the lifted theory that extend those template models without the assumption that multiplicities are one, to solve the problem for larger numbers of golfers and foursomes, as in Figure 4.13.

4.5 Discussion

Ideally, any compression of a concrete model of a sentence (Section 4.3.1 on page 70) should be a model of the lifted sentence. It is unlikely that this is achievable, though: the translation would have to require the satisfaction of only the regularity constraints for functions. This requires being able to predict the fraction of concrete variable assignments (in the expansion of a lifted assignment) that make a concrete formula true, given that the translated formula is true with the lifted assignment.

In particular, filter formulas (in an aggregate) having both free and quantified variables are problematic. The concept of seeds may be useful to transform them with higher compression. A *set of seeds for a tuple of lifted domain elements* is a minimal set of tuples of concrete domain elements such that the union of the π-orbits of the concrete tuple is the strong expansion of the lifted tuple.

Example 39. *Consider a lifted domain with two types:* `X≜{a}` *and* `Y≜{b}`*, where* `mul(a)=2=mul(b)`*. The expansion of the lifted domain is* $\{a^1, a^2, b^1, b^2\}$*, with permutation* $\pi = (a^1a^2)(b^1b^2)$*. The strong expansion of tuple* (a, b) *is* $\{(a^1, b^1), (a^1, b^2), (a^2, b^1), (a^2, b^2)\}$*. A set of seeds for the tuple is* $\{(a^1, b^1), (a^1, b^2)\}$*: the union of their* π*-orbits is the strong expansion of* (a, b)*.*

The value of an expression when a seed $\overline{sd}$ is assigned to a tuple of variables is preserved when the permutation is applied to the seed: $\phi^{I[\bar{x}:\overline{sd}]} = \phi^{I[\bar{x}:\pi^i(\overline{sd})]}$. Hence, the value of a sum aggregate for some variable assignments can be

computed as the sum computed over the seeds of the variable assignments, multiplied by an appropriate factor.

Example 40. *In our example,* $\sum_{x\in X, y\in Y} t(x,y) = 2 \times (t(a^1,b^1) + t(a^1,b^2))$.

This observation could be used to transform sum aggregates more efficiently. A challenge is to find a general method to determine the seeds of a tuple of lifted domain elements.

Lifted domain elements represent disjoint sets of concrete domain elements. Hence, a model search in the lifted domain can be seen as a model search involving sets. Hence, our work is highly related to Boolean Algebra of sets with Presburger Arithmetic (BAPA), a logic that can express constraints on the cardinality of sets, of their unions and of their intersections (Kuncak & Rinard, 2007; Suter et al., 2011; Bansal et al., 2016). Some problems from verification of properties of software operating on complex data structures contain fragments that belong to BAPA.

A sample BAPA statement is $|A| > 1 \wedge A \subseteq B \wedge |B \cap C| \leqslant 2$, where A, B, C are sets, and $|A|$ is the cardinality of A. The equivalent expression in FO(Type, Aggregate) is $(\#\{d : A(d)\} > 1) \wedge (\forall d : A(d) \Rightarrow B(d)) \wedge (\#\{d : B(d) \wedge C(d)\} \leqslant 2)$, where A, B, C are now unary predicates over a (unique) type whose interpretation is to be found. This expression can be lifted and solved using our approach (see example in Supplementary material[11]).

In general, any BAPA sentence can be converted to a concrete FO(Type, Aggregate) sentence that only uses unary predicates, and that can be lifted without regularity constraints. Hence, our approach offers a simple way to solve BAPA problems using any solver capable of reasoning over the rationals. The performance advantage should be evaluated in future work.

On the other hand, the conversion of a concrete FO(Type, Aggregate) sentence to BAPA logic is challenging because FO(Type, Aggregate) is more expressive: it allows n-ary relations, functions, sum aggregates, and product of cardinalities. Thus, a BAPA solver could not be used to solve, e.g., generative configuration problems. Still, extensions of BAPA solvers to handle finite n-ary relations have been implemented in CVC4 (Meng et al., 2017).

A simple approach to represent structures in BAPA is to use disjoint subsets of the concrete domain, called *Venn regions*, so that the cardinalities of any set of interest is the sum of the cardinalities of its Venn regions. Unfortunately, the number of Venn regions grows exponentially with the number of sets of interest. Hence, various methods have been developed to reduce this growth, e.g., by

[11] `https://gitlab.com/pierre.carbonnelle/idp-z3-generative`

creating new Venn regions lazily when required (Bansal et al., 2016). Venn regions are similar to our lifted domain elements, and the iterative method that creates new Venn region is similar, to some extent, to our iterative method that creates new lifted elements when required.

4.6 Personal contribution

This chapter is a significantly improved version of Carbonnelle et al. (2023a), submitted to AAAI 2024. I alone conceived the 3-step method to solve generative configuration problems in the lifted domain using a simple translation process, proved the equisatisfiability of the original and lifted sentences, and experimentally showed performance improvements on GCP problems from the literature (using only symbols with arity one). Gottfried Schenner created the experiment using CMPpy. Prof. Bruynooghe improved the clarity of the presentation of the method, and significantly helped improve the compression of theories with n-ary symbols. I further improved the method, wrote the final proof of equisatisfiability, and developed the solution of the Social Golfer problem.

Chapter 5

Conclusion

This chapter discusses future research directions and two possible applications of our results.

5.1 Future research directions

Although model expansion can solve many practical, real-world problems, more complex problems involving competing agents also occur frequently. Applying the knowledge base methodology to solve them is an interesting area of research. Often, probabilistic reasoning is required. Prior work includes CP-logic (Vennekens et al., 2009) and ProbLog (De Raedt et al., 2007). Support of transcendental functions (sin, cos) would also open up new applications in, e.g, computer aided architectural design.

As discussed in Section 4.2 on page 68, reformulating a model expansion problem can improve solving performance in the presence of symmetries, but finding an appropriate reformulation is more an art than a science. Turning it into a science, i.e., finding a systematic way to reformulate a problem for better performance, would be beneficial.

5.2 Application: Intelligent spreadsheets

Thanks to its flexibility, Excel is used by most companies to help make better decisions. Essentially, Excel is a model expansion engine: you give it data and

formulas, and it computes new data from the given data so that it satisfies the formulas.

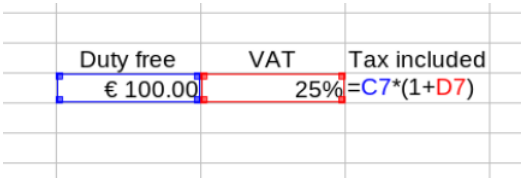


Figure 5.1: VAT calculation in Excel

While Excel performs computations very fast, its reasoning capabilities are rather limited. Consider the problem class of computing the tax-included amount, given the tax-excluded amount and the tax rate. Excel can be easily configured to solve instances of this problem class: two cells are used for the inputs, and a third cell for the output; the third cell has the formula to compute the tax-included amount (Figure 5.1). Now consider the different problem class of computing the tax-excluded amount, given the tax-included amount and the tax rate: you need to use three new cells, and enter a *new* formula to compute the tax-excluded amount. Yet, essentially, the *same* business formula applies.

This example illustrates again the difference between programming and knowledge representation (Section 2.1 on page 11). A program is written to perform one type of computation, i.e., to solve instances of *one* problem class in a problem domain, while a knowledge base can be used to solve instances of *any* problem class in the problem domain, with the appropriate reasoning engine.

This begs the question: could spreadsheets be made more intelligent? Could they use the knowledge base paradigm, instead of the more limited programming paradigm, to solve a wider variety of problems? It would allow you to enter the business formula linking the tax-excluded amount, tax rate and tax-included amount, and compute any one of these variables, given the other two. For large spreadsheets used in, among other things, budgeting, this flexibility could be very useful.

5.3 Application: Intelligent chatbots

Recently, chatGPT has taken the world by storm: it set the record for the fastest growing user base, reaching 100 million monthly active users just two months after its launch. Its conversational abilities bring us closer to a machine

passing the holy grail of AI research, the Turing test, i.e., to a machine whose conversation is indistinguishable from a conversation with a human.

Yet, the reasoning capabilities of chatGPT are rather limited: it cannot solve simple logic puzzles (Goossens et al., 2023). Combining conversational and reasoning capabilities in so-called hybrid systems is a hot topic of research, as it would yield intelligent machines with an easy-to-use user interface. Knowledge representation and automated reasoning have an important role to play in this endeavor. An attractive architecture would use neural networks to translate English sentences to FO($\cdot$) sentences that can be fed to a reasoning engine; the reasoning engine would perform precise inferences on this knowledge to prepare a formal response; the formal response would then be translated to English. Such an approach has been successfully used with the ProbLog and ASP languages (Suster et al., 2021; Ishay et al., 2023).

Appendix A

Partial functions

In this appendix, we describe how one can verify that an FO(·) theory is well-defined, i.e., that non-denoting terms appear in its sentences only in a context where they are irrelevant (see "Partial functions" in Section 2.2 on page 16). We also describe how to check that terms appear in contexts where they make sense, i.e., that sentences do not have "category clashes". Such verifications are done, for example, for the value function `$(.)` to verify that it is applied to the appropriate number of arguments of appropriate types (see "`Concept` as a type" in Section 2.2 on page 16).

The first act in Knowledge Representation is to identify the relevant concepts in a problem domain and to choose symbols to represent them. We observe that it is natural to identify concepts of two different kinds:

- categories (aka natural kinds), represented by type symbols;
- relations and properties, represented by predicate and function symbols.

An element in the domain of discourse is in categories that are intrinsic to its nature: in other words, one cannot imagine the element in another category than the ones he is in. For example, the element named *Alice* is a person and a living being, not a chair. We say that "person" is a category, represented by a type symbol, and that *Alice* is in that category.

Relations and properties are between objects of specific categories: for example, one talks about the spouse of a person, but not about the spouse of a chair. We say that the interpretation of *spouseOf* is a partial function whose argument

must be a person, and whose value for a person is a person. This information is represented by a *type signature*.

Finally, a function may not be totally defined on its type/category and hence, may be non-denoting for some objects in the appropriate category. For example, the spouse of *Alice* does not exist if *Alice* is not married, even though *Alice* is a person. In this case, the domain of the *spouseOf* function is the set of married persons. Due to the symmetry of the spouse relation, the set of married persons is also the codomain of this function. In the approach that we propose, for such partial functions, the user needs to introduce a predicate and declare it to be the domain of the partial function in a *domain signature*, and also its codomain.

So, when parsing logic sentences representing knowledge, we want to identify two types of errors:

- type errors (aka category clashes): when an argument of a symbol has an incorrect category / type;
- domain errors: when a possibly non-denoting term appears in a sentence in a context where it is relevant, i.e., when there is a structure in which the truth value of the sentence is undefined.

Example 41. *For example,* $\exists x, y : \frac{x}{y-1} > 1$ *has a domain error because a division by* 0 *is non-denoting, and this division appears in a context where it is relevant: the truth value of the sentence is not defined by the usual rules of interpretation in any structure that has the usual interpretations of the arithmetic operators.*

A domain error can be corrected by introducing *guard(s)* in the sentence and using a non-commutative (three-valued) interpretation of connectives. A corrected sentence for our example is: $\exists x, y : y \neq 1 \wedge \frac{x}{y-1} > 1$, in which the guard is $y \neq 1$. A non-commutative conjunction is used, e.g., in mathematics: $\psi \wedge \phi$ has the value of ϕ if ψ is true, is false if ψ is false, and is undefined otherwise. Notice that the corrected formula has no undefined instances since the guard is always true or false, and if the guard is true, then the division term is denoting.

We formalize these ideas below. We refine typed first-order logic (aka sorted logic, FO(T)) with subtypes, a subset relation, and partial functions; we call the result FO(PF).

Definition 30 (Typed, domained vocabulary)**.** *A* typed vocabulary Σ*, in FO(T), is a list of symbol declarations having one of the following forms:*

- **`type` `U`**, **`type` $\mathbb{B}$** , **`type` `Int`** *(predefined base types)*

- **type** T *(custom base types);*
- $p\colon T1\times..\times Tn \to \mathbb{B}$
  where $T1, .., Tn$ *are type symbols declared earlier in the list of declarations;*
- $f\colon T1\times..\times Tn \to T$
  where $T1, .., Tn, T$ *are type symbols declared earlier;*
- **var** $x \in T$*, where* T *is a type symbol declared earlier;*
- **var** $x \in p$*, where* p *is a predicate symbol declared earlier;*

where $T1\times..\times Tn \to T$ *is the* type signature *of* p *or* f*.*

A domained vocabulary*, in FO(PF), is a list of symbol declarations having one of the forms above and below:*

- **type** $T \sqsubseteq T1$
  where $T1$ *is a type declared earlier in the list (This ensures that the declared subtype relation is a finite partial order);*
- $p\colon T1\times..\times Tn \to \mathbb{B}$ $(p \subseteq q)$ *where* $T1, .., Tn$ *are type symbols, and* q *is a predicate symbol declared earlier with the same type signature as* p*;*
- $p\colon T1\times..\times Tn \to \mathbb{B}$ $(p \subseteq q1\times..\times qn)$
  where $T1, .., Tn$ *are type symbols, and each* qi *is either a type* Ti *or a unary predicate symbol declared earlier with type signature* $qi\colon Ti \to \mathbb{B}$*;*
- $f\colon T1\times..\times Tn \to T$ $(p \to q)$
  where $T1, .., Tn, T$ *are type symbols, and* p, q *are predicate symbols declared earlier with type signature* $T1\times..\times Tn \to \mathbb{B}$ *and* $T \to \mathbb{B}$ *respectively;*
- $f\colon T1\times..\times Tn \to T$ $(p1\times..\times pn \to q)$
  where $T1, .., Tn, T$ *are type symbols, and* $p1, .. pn, q$ *are predicate symbols declared earlier with type signature* $Ti \to \mathbb{B}$ *and* $T \to \mathbb{B}$ *respectively;*

where f*, and* $(p \to q)$ *is the* domain signature *of* f*. The meaning of the domain signature* $(p \to q)$ *of* f *is that its domain is the interpretation of* p*, and its codomain is the interpretation of* q*.*

The syntax of vocabularies in the IDP-Z3 implementation differs slightly from the above definition for the declaration of partial functions:

- the declaration of a partial function must be preceded with the `partial` keyword;
- the domain signature `(p → q)` is noted `(domain: p → codomain: q)` for better clarity;
- it is optional to specify the codomain of a partial function. If omitted, the codomain in the type signature (`T`) is used.
- it is optional to specify the domain signature of a partial function. If omitted, a predicate $\texttt{dom}_f$ is automatically declared for the domain of `f`. Thus, the declaration `partial f:T1 ×..×Tn→T` is syntactic sugar for:

$$\texttt{dom}_f\texttt{: T1 × .. × Tn} \to \mathbb{B}$$
$$\texttt{f: T1 × .. × Tn→T (domain: dom}_f\texttt{, codomain: T)}$$

Example 42. *The following list of declarations is a domained vocabulary used to describe a state of affairs about living beings and persons. The types are* `Int, LivingBeing` *and* `Person`*. Function* `weight` *is defined for all living beings. Function* `spouseOf` *is defined only for married persons, and its codomain is the set of married persons. It also declares variables p (for persons) and m (for married).*

```
type Int
type LivingBeing
type Person ⊑ LivingBeing

married : Person →  𝔹
is_wife : Person →  𝔹  (is_wife ⊆ married)

total   weight : LivingBeing → Int
partial spouseOf: Person → Person
                  (domain: married, codomain: married)
partial husbandOf: Person → Person
                  (domain: is_wife, codomain: married)

var p ∈ Person
var m ∈ married
```

Informally, a *typed structure* (resp. a *domained structure*) respects the type (resp. domain) signatures. Formally:

Definition 31 (Typed, domained structure)**.** *A* typed structure I *for typed vocabulary* Σ *consists of a nonempty set, called the domain of discourse and denoted* D*, and a mapping from each symbol* σ *in* Σ *to its interpretation* σ^I*:*

- *`U` is interpreted by D, $\mathbb{B}$ by $\{\textbf{\textit{true}}, \textbf{\textit{false}}\}$, and `Int` by $\mathbb{Z}$.*
- *Type `T`, declared by `type T`, is interpreted by a subset of D, disjoint from the other sets interpreting base types.*
- *Predicate `p`, declared by `p: T1×..×Tn → 𝔹`, is interpreted by a total boolean function over the cross-product $\texttt{T1}^I \times .. \times \texttt{Tn}^I$.*
- *Function `f`, declared by `f: T1×..×Tn → T`, is interpreted by a total function over the cross-product $\texttt{T1}^I \times .. \times \texttt{Tn}^I$, with codomain `T`.*

A domained structure *for domained vocabulary Σ also consists of a domain D and a mapping from each symbol σ in Σ to its interpretation σ^I, defined by the above and below:*

- *Type `T`, declared by `type T ⊑ T1`, is interpreted by a subset of $\texttt{T1}^I$.*
- *Predicate `p`, declared by `p: T1×..×Tn → 𝔹 (p ⊆ q)`, is interpreted by a total boolean function over the cross-product $\texttt{T1}^I \times .. \times \texttt{Tn}^I$ such that $\forall \bar{x} \in \bar{T}^I : p^I(\bar{x}) \Rightarrow q^I(\bar{x})$ is true.*
- *Predicate `p`, declared by `p: T1×..×Tn → 𝔹 (p ⊆ q1×..×qn)`, is interpreted by a total boolean function over the cross-product $\texttt{T1}^I \times .. \times \texttt{Tn}^I$ such that $\forall x_1 \ldots x_n \in \bar{T}^I : p^I(x_1, \ldots, x_n) \Rightarrow q_1^I(x_1) \wedge \cdots \wedge q_n^I(x_n)$ is true.*
- *Function `f`, declared by `f: T1×..×Tn → T (p → q)`, is interpreted by a partial function over the cross-product $\texttt{T1}^I \times .. \times \texttt{Tn}^I$, whose domain of definition is exactly $\{\bar{x} \in \bar{T}^I \mid p^I(\bar{x}) = \textbf{\textit{true}}\}$ and whose image is a subset of $\{y \in T^I \mid q^I(y) = \textbf{\textit{true}}\}$.*
- *Function `f`, declared by `f: T1×..×Tn → T (p1×..×pn → q)`, is interpreted by a partial function over the cross-product $\texttt{T1}^I \times .. \times \texttt{Tn}^I$, whose domain of definition is exactly $\{\bar{x} \in \bar{T}^I \mid \forall i : p_i^I(x_i) = \textbf{\textit{true}}\}$ and whose image is a subset of $\{y \in T^I \mid q^I(y) = \textbf{\textit{true}}\}$.*

Example 43. *The following structure is a domained structure for the vocabulary in Example 42 on the facing page.*

```
livingBeing ≜ {Winnie, Bob, Alice, Tom}
Person ≜ {Bob, Alice, Tom}

married ≜ {Bob→true, Alice→true, Tom→false}
is_wife ≜ {Bob→false, Alice→true, Tom→false}

weight ≜ {Winnie→6, Bob→83, Alice→75, Tom→90}
spouseOf ≜ {Bob→Alice, Alice→Bob}
husbandOf ≜ {Alice → Bob}
```

It would not be a domained structure if the interpretation of `husbandOf` *was* `husbandOf` $\triangleq$ `{Tom` $\rightarrow$ `Tom}` *because* `Tom` *is not in its domain, i.e., is not in* `is_wife`.

Definition 32 (Extended typed, domained structure)**.** *An* extended typed structure *(resp. domained structure) for typed (resp. domained) vocabulary* Σ*, denoted* $I[\bar{x} : \bar{v}]$*, is a typed (resp. domained) structure for* Σ *where each variable symbol* x_i *is given a value according to its declaration in* Σ*:*

- *Variable* `x`*, declared by* `var` $x_i \in$ `T`*, is interpreted by an element of* T^I
- *Variable* `x`*, declared by* `var` $x_i \in$ `p` *(with* `p:T`$\rightarrow \mathbb{B}$ *), is interpreted by an element of* T^I *such that* $p^I(x_i^I) =$ ***true****.*

Definition 33 (Subtype relation)**.** *The subtype relation* $\sqsubseteq^*$ *between type symbols declared in a domained vocabulary is a partial order for the set of type symbols, defined inductively as follows:*

- $T \sqsubseteq^* T$ *for all type symbols;*
- $T \sqsubseteq^* T_2$ *when* T *is declared by* `type T` $\sqsubseteq$ `T1` *and when* $T_1 \sqsubseteq^* T_2$*;*

Definition 34 (Type of an expression)**.** *The type of a* Σ*-expression* ϕ *is denoted by* $\tau(\phi)$ *and is inductively defined as follows:*

- $\tau(x_i) = T$ *if* x *is declared by* `var x` $\in$ `T` *or* `var x` $\in$ `p` *with* `p:T`$\rightarrow \mathbb{B}$
- $\tau(T(t)) = \mathbb{B}$*;*
- $\tau(p(\bar{t})) = \mathbb{B}$ *if* p *has type signature* $\bar{T} \rightarrow \mathbb{B}$*, and* $\tau(t_i) \sqsubseteq^* T_i$ *for each* i*;*
- $\tau(f(\bar{t})) = T$ *if* f *has type signature* $\bar{T} \rightarrow T$*, and* $\tau(t_i) \sqsubseteq^* T_i$ *for each* i*;*
- $\tau(\neg(\psi)) = \mathbb{B}$ *if* $\tau(\psi) = \mathbb{B}$*;*
- $\tau(\psi_1 \vee \psi_2) = \mathbb{B}$ *if* $\tau(\psi_1) = \tau(\psi_2) = \mathbb{B}$*.*
- $\tau(\exists y : \psi) = \mathbb{B}$ *if* y *is declared in the vocabulary, and* $\tau(\psi) = \mathbb{B}$*.*

and is undefined otherwise.

The definition above can be extended to other language constructs.

Definition 35 (Well-typed expression)**.** *An expression* ϕ *is* well-typed *if its type* $\tau(\phi)$ *is not undefined.*

Example 44. *Continuing our example,* `married(Bob)` *is well-typed, but* `married(60)` *is not:* 60 *is not a* `Person` *(*`Int` *is not a subtype of* `Person`*).*

Definition 36 (Value of a well-typed expression)**.** *The value of a well-typed expression ϕ in extended domained structure $I^* = I[\bar{x} : \bar{v}]$ is inductively defined over the structure of the expression. It is:*

- v_i *if* ϕ *is* x_i*;*
- $(t^{I^*} \in T^{I^*})$ *if* ϕ *is* $T(t)$ *and* t^{I^*} *is defined;*
- $p^{I^*}(\bar{t}^{I^*})$ *if* ϕ *is* $p(\bar{t})$ *and* p *is a predicate symbol and* t^{I^*} *is defined;*
- $f^{I^*}(\bar{t}^{I^*})$ *if* ϕ *is* $f(\bar{t})$ *and* f *is a function symbol and* $\bar{t}^{I^*}$ *is defined and in the domain of definition of* f*;*
- $\neg(\psi^{I^*})$ *if* ϕ *is* $\neg(\psi)$ *if* ψ^{I^*} *is defined;*
- ***true*** *if* ϕ *is* $\psi_1 \vee \psi_2$ *and* $\psi_1^{I^*}$ *is true;*
- $\psi_2^{I^*}$ *if* ϕ *is* $\psi_1 \vee \psi_2$ *and* $\psi_1^{I^*}$ *is false;*
- $\bigvee_{v_y \in T^{I^*}} (\psi^{I[\bar{x}:\bar{v}::y:v_y]})$ *if* ϕ *is* $\exists y : \psi$ *and* y *is declared as* `var y ∈ T` *and* $\psi^{I[\bar{x}:\bar{v}::y:v_y]})$ *is defined for any* v_y *in* T^{I^*}*.*
- $\bigvee_{v_y \in T^{I^*}} (q^{I^*} \Rightarrow \psi^{I[\bar{x}:\bar{v}::y:v_y]})$ *if* ϕ *is* $\exists y : \psi$ *and* y *is declared as* `var y ∈ q` *with* `q⊆T`*, and* $(q^{I^*} \Rightarrow \psi^{I[\bar{x}:\bar{v}::y:v_y]})$ *is defined for any* v_y *in* T^{I^*}*.*

The value is undefined otherwise.

Notice that the rule for the disjunction is non-commutative.

Definition 37 (Well-domained expression)**.** *A well-typed expression ϕ is* well-domained *in extended domained structure $I^* = I[\bar{x} : \bar{v}]$ iff it is of the form:*

- x_i
- $T(t)$ *where* T *is a type symbol and* t *is well-domained in* I^**;*
- $p(\bar{t})$ *where* p *has type signature* $\bar{T} \to T$*, and* $\bar{t}$ *(of type* $\bar{T}$*) is well-domained in* I^**;*
- $f(\bar{t})$ *where* f *has signature* $\bar{T} \to T$ $(p \to q)$*,* $\bar{t}$ *(of type* $\bar{T}$*) is well-domained in* I^**, and* $p(\bar{t})$ *is true in* I^**;*

- $f(\bar{t})$ *where* f *has signature* $\bar{T} \to T$ $(p_1 \times \ldots \times p_n \to q)$, $\bar{t}$ *(of type* $\bar{T}$*) is well-domained in* I^**, and* $p_i(t_i)$ *is true in* I^* *for all* i*;*
- $\neg(\psi)$ *where* ψ *is a well-domained expression in* I^**;*
- $\psi_1 \vee \psi_2$ *where* ψ_1 *is a well-domained expression in* I^**, and either* ψ_1 *is true in* I^* *or* ψ_2 *is well-domained in* I^**.*
- $\exists y : \psi$ *where* y *is declared as* `var y ∈ T` *(resp.* `var y ∈ q` *with* `q⊆T`*) and* ψ *is a well-domained expression in* $I[\bar{x} : \bar{v} :: y : v_y]$ *for each* v_y *in* T^{I^*} *(resp. in* $\{y \in T^{I^*} \mid q^{I^*}(y)\}$*).*

Theorem 2. *An expression that is well-domained in extended structure* I^* *has a defined value in* I^**.*

Proof. The property is proven inductively over the structure of the expression. In particular, the condition for a term to be well-domained in I^* ensures that it has a value in I^*. The inductive step is: "if each subexpression of ϕ has a value in I^* and if ϕ is well-domained in I^*, then ϕ also has a value in I^*". □

Notice that the evaluation of a disjunction is asymmetric and non-strict.

A *well-defined* sentence has a value in any domained structure. Formally:

Definition 38 (Well-defined formula)**.** *A sentence over a domained vocabulary* Σ *is* well-defined *iff it is well-domained in any domained* Σ*-structure.*

Example 45. *Sentence* $a() = 0 \vee b() = 1/a()$*, in a vocabulary where* a *and* b *are declared as* $a, b : () \to \mathbb{R}$*, is well-typed and well-defined: it has a value in any domained structure that has the usual interpretation of equality and division (and the non-commutative interpretation of disjunction). On the other hand,* $b() = 1/a() \vee a() = 0$ *is well-typed but not well-defined: it is not well-domained and it has no value when the interpretation of* $a()$ *is* 0*.*

Informally, the T-extension of a domained structure I is a typed structure obtained by arbitrarily extending the interpretation of the partial functions in I. Formally:

Definition 39. *A* T-extension *of domained* Σ*-structure* I *is a typed* Σ*-structure* J *that satisfies the following constraints:*

- $U^J = U^J, \mathbb{B}^J = \mathbb{B}^I, Int^J = Int^I$
- *for each type* T*,* $T^J = t^I$*;*

- *for each predicate* p, $p^J = p^I$;
- *for each total function* f, $f^J = f^I$;
- *for each function* $f : \bar{T} \times T(p \to q)$, f^J *is a total function over* $\bar{T}^I$ *such that* $\forall \bar{d} \in \bar{T}^I : p(\bar{d}) \Rightarrow f^J(\bar{d}) = f^I(\bar{d})$.
- *for each function* $f : \bar{T} \times T(p_1 \times \ldots \times p_n \to q)$, f^J *is a total function over* $\bar{T}^I$ *such that* $\forall \bar{d} \in \bar{T}^I : (\forall i : p_i(d_i)) \Rightarrow f^J(\bar{d}) = f^I(\bar{d})$.

Example 46. *Below is a T-extension of the structure of Example 43 on page 123:*

```
livingBeing ≜ {Winnie, Bob, Alice, Tom}
Person ≜ {Bob, Alice, Tom}

married ≜ {Bob→true, Alice→true, Tom→false,
           Winnie→true}
is_wife ≜ {Bob→false, Alice→true, Tom→false,
           Winnie→true}

weight ≜ {Winnie→6, Bob→83, Alice→75, Tom→90}
spouseOf ≜ {Bob→Alice, Alice→Bob,
            Winnie→Winnie, Tom→Tom}
husbandOf ≜ {Alice → Bob
             Winnie→Winnie, Bob→Bob, Tom→Tom}
```

Thus, the interpretation of functions in a T-extension is always total.

Theorem 3. *A sentence is well-domained in* I *iff it is well-typed and it has the same value in any T-extension of* I*; that common value is the same as the value in* I.

Proof. The interpretation of a well-domained expression in I per Definition 36, and in a T-extension of I per the interpretation rules by typed logic, differ only for partial functions. The interpretations of partial functions differ only outside their domain, but the well-domained condition ensures that values of functions outside their domain are irrelevant when determining the truth value of the sentence. □

Observe that the condition for an expression to be well-domained (Definition 37) only depends on the interpretation of total predicate symbols, and not on the interpretation of function symbols: the sentence representing it (see Definition 40 on the following page) is thus well-domained in any domained structure I over

the same vocabulary. By Theorem 3, its value in any T-extension of I is its value in I. We use this property to create the method.

The method to verify that a given well-typed Σ-sentence is well-defined consists of (i) representing the well-domained condition for that sentence by a formula ψ in typed logic, and (ii) verifying that ψ is true in any T-extension of any well-domained structure. By Theorem 3, if it is true in any T-extension, it is also true in any well-domained structure, and the formula is well-defined.

Thus, for sentence ϕ to be well-defined, the following sentence must be true in any typed structure: "if the typed structure is a T-extension of a domained structure I, then the (well-typed) sentence ϕ is well-domained in I (and thus has a defined value in I)". The first part of the condition, denoted T_s, captures the knowledge in the domained vocabulary. The second part, denoted $\mathcal{D}_\phi$, is obtained by translation of the sentence according to the definition of a well-domained expression (Definition 37). Sentence ϕ is well-defined iff the well-definedness condition $T_s \Rightarrow \mathcal{D}_\phi$ is true in any typed structure (i.e., iff its negation is unsatisfiable).

The first part, T_s, is the conjuction of the sentences listed in Figure A.1, some of which use binary quantification.

	T_s
for each *type* $T \sqsubseteq T_1$	$\forall x \in T : T_1(x)$.
for each $p : T \to \mathbb{B}$ $(p \subseteq p_1)$	$\forall \bar{x} \in p : p_1(\bar{x})$
for each $p : \bar{T} \to \mathbb{B}$ $(p \subseteq p_1 \times \ldots \times p_n)$	$\forall \bar{x} \in p : \forall i : p_i(x_i)$
for each $f : \bar{T} \to T$ $(p \to q)$	$\forall \bar{x} \in p : q(f(\bar{x}))$.
for each $f : \bar{T} \to T$ $(p_1 \times \ldots \times p_n \to q)$	$\forall \bar{x} \in \bar{T} : (\forall i : p_i(x_i)) \Rightarrow q(f(\bar{x}))$.

Table A.1: Knowledge in the domain signatures

The second part, $\mathcal{D}_\phi$, is defined below, following Berezin et al. (2004):

Definition 40 (Well-Domained Condition)**.** *Let ϕ be an arbitrary well-typed expression, the* ***well-domained condition*** *formula $\mathcal{D}_\phi$ is defined inductively*

by these rules:

$$\mathcal{D}_x \equiv true \tag{A.1}$$

$$\mathcal{D}_{f(t_1,\ldots,t_n)} \equiv \mathcal{D}_{t_1} \wedge \cdots \wedge \mathcal{D}_{t_n} \wedge p_1(t_1,\ldots,t_n)^1 \tag{A.2}$$

$$\mathcal{D}_{\neg\psi} \equiv \mathcal{D}_{\psi} \tag{A.3}$$

$$\mathcal{D}_{\psi_1 \vee \psi_2} \equiv \mathcal{D}_{\psi_1} \wedge (\psi_1 \vee \mathcal{D}_{\psi_2}) \tag{A.4}$$

$$\mathcal{D}_{\forall y \in q:\psi} \equiv \forall y \in q : \mathcal{D}_{\psi} \tag{A.5}$$

Again, such a definition can be extended to other language constructs.

Example 47. *We want to prove that, in the domained vocabulary of Example 42, the following sentence is well-defined :*

$$\forall m \in married : spouseOf(spouseOf(m)) = m$$

The relevant part of T_s is the conjunct for symbol spouseOf:

$$\forall m \in married : married(spouseOf(m)) \tag{A.6}$$

The well-domained condition for the formula is

$$\mathcal{D}_{\phi} = \forall m \in married : married(m) \wedge married(spouseOf(m)) \tag{A.7}$$

$$= \forall m \in married : married(spouseOf(m)) \tag{A.8}$$

Sentences T_s, $\mathcal{D}_{\phi}$ are identical, and the well-definedness condition $T_s \Rightarrow \mathcal{D}_{\phi}$ is indeed a tautology.

A.1 Personal contribution

Following extensive discussions with Prof. Denecker and Djordje Markovic, I conceived the concept of domained vocabulary (as distinct from typed vocabulary), wrote the description of FO(PF) in this chapter, and implemented it in IDP-Z3.

[1] where the signature of f is $f : \bar{T} \to T$ $(p_1 \to p_2)$.

Appendix B

Transformation to lift FO(Type, Aggregate) sentences

Section 4.3.2 on page 77 described how to lift FO(Type, Aggregate) sentences that are in normal form. In this chapter, we describe how to lift any FO(Type, Aggregate) sentence. The transformation is described in Table B.1 on the next page.

The proof of equisatisfiability for FO(Type, Aggregate) sentences is similar to the proof for FO(Type, Aggregate) sentences in normal form. The "preservation of truth value" property is extended as follows:

Proposition 12 (Preservation of truth values)**.**

- *(i) if* $W(\psi)$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any* i*,* ψ *is true in* $I[\bar{y} : \pi^i(\bar{l})]$*.*
- *(ii) if* $S(\psi)$ *is true in* $L[\bar{y} : \bar{l}]$*, then* ψ *is true in every instance of* $L[\bar{y} : \bar{l}]$*.*
- *(iii) If* $W^F(\psi)$ *is true in* $L[\bar{y} : \bar{l}]$*, then, for any* i*,* ψ *is false in* $I[\bar{y} : \pi^i(\bar{l})]$*.*
- *(iv) If* $S^F(\psi)$ *is true in* $L[\bar{y} : \bar{l}]$*, then* ψ *is false in every instance of* $L[\bar{y} : \bar{l}]$*.*

Rule	Transformations			
ϕ	$W(\phi)$	$S(\phi)$	$W^F(\phi)$	$S^F(\phi)$
true/false	**true/false**	**true/false**	**false/true**	**false/true**
$p()$	$p()$	$p()$	$\neg p()$	$\neg p()$
$p(\bar{t})$	$p(\chi(\bar{t})) \wedge R(\bar{t})$	$p(\chi(\bar{t})) \wedge Reg(\bar{t})$	$\neg p(\chi(\bar{t})) \wedge R(\bar{t})$	$\neg p(\chi(\bar{t})) \wedge R(\bar{t})$
$\psi_1 \wedge \psi_2$	$W(\psi_1) \wedge W(\psi_2)$	$S(\psi_1) \wedge S(\psi_2)$	$W^F(\psi_1) \vee W^F(\psi_2)$	$S^F(\psi_1) \vee S^F(\psi_2)$
$\psi_1 \vee \psi_2$	$W(\psi_1) \vee W(\psi_2)$	$S(\psi_1) \vee S(\psi_2)$	$W^F(\psi_j) \wedge W^F(\psi_j)$	$S^F(\psi_j) \wedge S^F(\psi_j)$
$\neg\psi$	$W^F(\psi)$	$S^F(\psi)$	$W(\psi)$	$S(\psi)$
$T \triangleq \{t_1,..,t_n\}$	$\sum_{x \in T} mul(x) = n$			
$\forall \bar{x} \in \bar{T}^0 : \psi$	$\forall \bar{x} \in \bar{T}^0 : S(\psi)$	$\forall \bar{x} \in \bar{T}^0 : S(\psi)$	$\exists \bar{x} \in \bar{T}^0 : W^F(\psi)$	$S(\exists \bar{x} \in \bar{T} : \neg\psi)$
$\exists \bar{x} \in \bar{T}^0 : \psi$	$\exists \bar{x} \in \bar{T}^0 : W(\psi)$	$\exists \bar{x} \in \bar{T}^0 : S(\psi) \vee [Reg(\bar{y}) \wedge W(\psi)]$	$\forall \bar{x} \in \bar{T}^0 : S^F(\psi)$	$\forall \bar{x} \in \bar{T}^0 : S^F(\psi)$
t	$w(t)$	$s(t)$	$\chi(t)$	
x	**true**	**true**	x	
$c()$	**true**	**true**	$c()$	
$f(\bar{t})$	$w(\bar{t})$	$s(\bar{t})$	$f(\chi(\bar{t}))$	
$\sum_{\bar{x} \in \bar{T} : \psi} t_0$	$\forall \bar{x} \in \bar{T}^0 : s(t_0) \wedge (S(\psi) \vee S(NF(\neg\psi)))$	$\forall \bar{x} \in \bar{T}^0 : s(t_0) \wedge (S(\psi) \vee S(NF(\neg\psi)))$	$\sum_{\bar{x} \in \bar{T}^0 : S(\psi)} Mul(\bar{x}) \times \chi(t_0)$	

Table B.1: Transformation of formulas and terms

Appendix C

Lifted theory for the Social Golfer problem

Listing C.1: Lifted theory for the 8-4-2 Social Golfer problem

```
vocabulary V{
  type G_T
  type W
  type FS_T

  mulG : G_T → Int  // multiplicity
  mulFS : FS_T → Int

  G : G_T →  𝔹   // golfers with multiplicity > 0
  FS : FS_T →  𝔹

  f,s,t,ff: FS → G
  week : FS → W
  nr_plays: G × G → Int

  var g in G
  var fs in FS
  var w in W
}
theory T:V {

  // T'
  ∀ x ∈ G_T: G(x) ⇔ 0 < mulG(x).
  ∀ x ∈ FS_T: FS(x) ⇔ 0 < mulFS(x).
```

```
  // Simplifying constraint
  ∀x ∈ FS_T: 0 ⩽ mulFS(x) = mulG(f(x)) = mulG(s(x))
                          = mulG(t(x)) = mulG(ff(x)).

  // Golfers in a foursome are different + symmetry breaking
  ∀ fs: 0 < f(fs) < s(fs) < t(fs) < ff(fs).

  // Each golfer plays once a week.
  ∀ g, w: #{fs : week(fs) = w ∧  f(fs) = g}
        + #{fs : week(fs) = w ∧  s(fs) = g}
        + #{fs : week(fs) = w ∧  t(fs) = g}
        + #{fs : week(fs) = w ∧ ff(fs) = g}
        = 1.

  // a golfer does not play more than once with another.
  ∀ g1, g2: #{fs :
        (g1=f(fs) ∧ g2 ∈ {s(fs), t(fs), ff(fs)})
      ∧ (g1=s(fs) ∧ g2 ∈ {f(fs), t(fs), ff(fs)})
      ∧ (g1=t(fs) ∧ g2 ∈ {f(fs), s(fs), ff(fs)})
      ∧ (g1=ff(fs) ∧ g2 ∈ {f(fs), s(fs), t(fs)}) } ⩽ 1.

  // first foursomes
  ∀ fs in FS_T: fs ⩽ #{x in G_T}/4 ⇒
      week(fs)=1
      ∧ f(fs)=(fs-1) × 4+1 ∧  s(fs)=(fs-1) × 4+2
      ∧ t(fs)=(fs-1) × 4+3 ∧ ff(fs)=(fs-1) × 4+4.

  // foursome in lexicographic order
  ∀ fs1, fs2: #{x in G_T}/4 < fs1 < fs2 ⇒
       week(fs1)<week(fs2)
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)<f(fs2))
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)=f(fs2)∧s(fs1)<s(fs2))
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)=f(fs2)∧s(fs1)=s(fs2)
            ∧t(fs1)<t(fs2))
    ∨ (week(fs1)=week(fs2) ∧ f(fs1)=f(fs2)∧s(fs1)=s(fs2)
            ∧t(fs1)=t(fs2) ∧ff(fs1)<ff(fs2)).

  // sizes of concrete types.
  W ≜ {1..2}                          // w (not lifted)
  sum{{mulG(g) | g ∈ G_T}} = 32.      // g*s
  sum{{mulFS(fs) | fs ∈ FS}} = 16.    // g*w
}
```

List of publications

Peer-reviewed paper:

- Carbonnelle, Pierre and Vennekens, Joost and Denecker, Marc and Bogaerts, Bart. "Interactive Model Expansion in an Observable Environment". In: *Theory and Practice of Logic Programming* 23.4 (2023), pp. 648–663. doi: 10.1017/S1471068423000261

Conference proceedings:

- Carbonnelle, Pierre and Schenner, Gottfried and Bruynooghe, Maurice and Bogaerts, Bart and Denecker, Marc. "Using symmetries to lift satifiability checking". In: *CoRR abs/2311.03424* (2023). arXiv: 2311.03424. Accepted for publication at AAAI'24.
- Carbonnelle, Pierre and Van der Hallen, Matthias and Denecker, Marc. "Quantification and aggregation over concepts of the ontology". In: Electronic Proceedings in Theoretical Computer Science 385, pp. 213-226 (2023), doi:10.4204/eptcs.385.22

Workshops

- Carbonnelle, Pierre and Vennekens, Joost and Denecker, Marc and Bogaerts, Bart. "Interactive advisor for lax legislation and observable situations". In: *Workshop on Models of Legal Reasoning*. Vol. 9. 2020
- Carbonnelle, Pierre and Aerts, Bram and Deryck, Marjolein and Vennekens, Joost and Denecker, Marc. "An Interactive Consultant". In: *BNAIC/BENE-LEARN*. 2019, vol. 2491 of CEUR Workshop Proceedings. CEUR-WS.org

ArXiv:

- Carbonnelle, Pierre and Vandevelde, Simon and Vennekens, Joost and Denecker, Marc. "IDP-Z3: a reasoning engine for FO(.)". In: *CoRR abs/2202.00343* (2022). arXiv: 2202.00343

Bibliography

Abiteboul, S., Bienvenu, M., Galland, A., & Antoine, É. (2011). A rule-based language for web data management. In M. Lenzerini, & T. Schwentick (Eds.) *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, (pp. 293–304). ACM.

Aerts, B., Deryck, M., & Vennekens, J. (2022). Knowledge-based decision support for machine component design: A case study. *Expert Systems with Applications*, *187*, 115869.
URL `https://www.sciencedirect.com/science/article/pii/S0957417421012288`

Aschinger, M., Drescher, C., Gottlob, G., & Vollmer, H. (2014). LoCo - A logic for configuration problems. *ACM Trans. Comput. Log.*, *15*(3), 20:1–20:25.

Axling, T., & Haridi, S. (1996). A tool for developing interactive configuration applications. *J. Log. Program.*, *26*(2), 147–168.
URL `https://doi.org/10.1016/0743-1066(95)00097-6`

Bansal, K., Reynolds, A., Barrett, C., & Tinelli, C. (2016). A new decision procedure for finite sets and cardinality constraints in SMT. In *International Joint Conference on Automated Reasoning*, (pp. 82–98). Springer.

Barker, V. E., & O'Connor, D. E. (1989). Expert systems for configuration at digital: XCON and beyond. *Commun. ACM*, *32*(3), 298–318.
URL `https://doi.org/10.1145/62065.62067`

Barklund, J., Acqua, P. D., Costantini, S., & Lanzarone, G. A. (2000). Reflection principles in computational logic.

Barrett, C., Stump, A., & Tinelli, C. (2010a). The SMT-LIB Standard: Version 2.0. In A. Gupta, & D. Kroening (Eds.) *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*.

Barrett, C., Stump, A., & Tinelli, C. (2010b). The SMT-LIB Standard: Version 2.0. In A. Gupta, & D. Kroening (Eds.) *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*.

Berezin, S., Barrett, C. W., Shikanian, I., Chechik, M., Gurfinkel, A., & Dill, D. L. (2004). A practical approach to partial functions in CVC lite. In W. Ahrendt, P. Baumgartner, H. de Nivelle, S. Ranise, & C. Tinelli (Eds.) *Selected Papers from the Workshops on Disproving, D@IJCAR 2004, and the Second International Workshop on Pragmatics of Decision Procedures, PDPAR@IJCAR 2004, Cork, Ireland, July 2004*, vol. 125 of *Electronic Notes in Theoretical Computer Science*, (pp. 13–23). Elsevier.
URL `https://doi.org/10.1016/j.entcs.2004.06.064`

Bjørner, N., Levatich, M., Lopes, N. P., Rybalchenko, A., & Vuppalapati, C. (2021). Supercharging plant configurations using Z3. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*, (pp. 1–25). Springer.

Bjørner, N. S., de Moura, L., Nachmanson, L., & Wintersteiger, C. M. (2018). Programming Z3. In J. P. Bowen, Z. Liu, & Z. Zhang (Eds.) *Engineering Trustworthy Software Systems - 4th International School, SETSS 2018, Chongqing, China, April 7-12, 2018, Tutorial Lectures*, vol. 11430 of *Lecture Notes in Computer Science*, (pp. 148–201). Springer.
URL `https://doi.org/10.1007/978-3-030-17601-3_4`

Bogaerts, B., Vennekens, J., & Denecker, M. (2018). Safe inductions and their applications in knowledge representation. *Artif. Intell.*, *259*, 167–185.
URL `https://doi.org/10.1016/j.artint.2018.03.008`

Bogaerts, B., Vennekens, J., Denecker, M., & Van den Bussche, J. (2014). FO(C): A knowledge representation language of causality. *arXiv preprint arXiv:1405.1833*.

Bowen, J., & Bahler, D. (1991). Conditional existence of variables in generalised constraint networks. In T. L. Dean, & K. R. McKeown (Eds.) *Proceedings of the 9th National Conference on Artificial Intelligence*, (pp. 215–220). AAAI Press.
URL `http://www.aaai.org/Library/AAAI/1991/aaai91-034.php`

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., & Schaub, T. (2012). ASP-Core-2: Input language format. *ASP Standardization Working Group*.

Carbonnelle, P., Aerts, B., Deryck, M., Vennekens, J., & Denecker, M. (2019). An interactive consultant. In K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts,

N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, & P. V. Eecke (Eds.) *Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (Benelearn 2019), Brussels, Belgium, November 6-8, 2019*, vol. 2491 of *CEUR Workshop Proceedings*. CEUR-WS.org.
URL https://ceur-ws.org/Vol-2491/demo45.pdf

Carbonnelle, P., Schenner, G., Bruynooghe, M., Bogaerts, B., & Denecker, M. (2023a). Using symmetries to lift satisfiability checking. *arXiv preprint arXiv:2311.03424*.

Carbonnelle, P., Van der Hallen, M., & Denecker, M. (2023b). Quantification and aggregation over concepts of the ontology. *Electronic Proceedings in Theoretical Computer Science*, *385*, 213–226.
URL https://doi.org/10.4204%2Feptcs.385.22

Carbonnelle, P., Vandevelde, S., Vennekens, J., & Denecker, M. (2022). IDP-Z3: a reasoning engine for FO(.). *CoRR*, *abs/2202.00343*.

Carbonnelle, P., Vennekens, J., Denecker, M., & Bogaerts, B. (2023c). Interactive model expansion in an observable environment. *Theory and Practice of Logic Programming*, *23*(4), 648–663.

Carbonnelle, P., Vennekens, J., Denecker, M., & Bogaerts, B. (2023d). Interactive model expansion in an observable environment. *Theory and Practice of Logic Programming*, *23*(4), 648–663.

Chen, W., Kifer, M., & Warren, D. S. (1993). HiLog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, *15*(3), 187–230.

Cimatti, A., Griggio, A., & Sebastiani, R. (2007). A simple and flexible way of computing small unsatisfiable cores in SAT Modulo Theories. In J. Marques-Silva, & K. A. Sakallah (Eds.) *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, vol. 4501 of *Lecture Notes in Computer Science*, (pp. 334–339). Springer.
URL https://doi.org/10.1007/978-3-540-72788-0_32

Clark, K. L. (1977). Negation as failure. In H. Gallaire, & J. Minker (Eds.) *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, (pp. 293–322). New York: Plemum Press.
URL https://doi.org/10.1007/978-1-4684-3384-5_11

Comploi-Taupe, R., Francescutto, G., & Schenner, G. (2022). Applying incremental answer set solving to product configuration. In A. Felfernig

et al. (Ed.) *SPLC '22: 26th ACM Int'l Systems and Software Product Line Conference*, (pp. 150–155). ACM.

Crawford, J. M., Ginsberg, M. L., Luks, E. M., & Roy, A. (1996). Symmetry-breaking predicates for search problems. In L. C. Aiello, J. Doyle, & S. C. Shapiro (Eds.) *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, (pp. 148–159). Morgan Kaufmann.

De Cat, B., Bogaerts, B., & Denecker, M. (2014). MiniSAT(ID) for satisfiability checking and constraint solving. *ALP Newsletter*.

De Cat, B., Jansen, J., & Janssens, G. (2013). IDP3: Combining symbolic and ground reasoning for model generation. In *2nd Workshop on Grounding and Transformations for Theories With Variables*, (pp. 17–24).

De Moura, L., & Bjørner, N. (2008). Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, (pp. 337–340). Springer.

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence*, (pp. 2462–2467). IJCAI-Int Joint Conf Artif Intell.

de Salvo Braz, R., Amir, E., & Roth, D. (2005). Lifted first-order probabilistic inference. In L. P. K. et al. (Ed.) *IJCAI-05, Proceedings of the Nineteenth Int'al Joint Conference on Artificial Intelligence, 2005*, (pp. 1319–1325). Professional Book Center.

Denecker, M. (2000). 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, & P. J. Stuckey (Eds.) *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, vol. 1861 of *Lecture Notes in Computer Science*, (pp. 703–717). Springer.
URL `https://doi.org/10.1007/3-540-44957-4_47`

Denecker, M., Brewka, G., & Strass, H. (2015). A formal theory of justifications. In F. Calimeri, G. Ianni, & M. Truszczynski (Eds.) *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, vol. 9345 of *Lecture Notes in Computer Science*, (pp. 250–264). Springer.
URL `https://doi.org/10.1007/978-3-319-23264-5_22`

Denecker, M., Bruynooghe, M., & Vennekens, J. (2012). Approximation fixpoint theory and the semantics of logic and answers set programs. In *Correct reasoning*, (pp. 178–194). Springer.

Denecker, M., Pelov, N., & Bruynooghe, M. (2001). Ultimate well-founded and stable semantics for logic programs with aggregates. In P. Codognet (Ed.) *Logic Programming, 17th International Conference, ICLP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, vol. 2237 of *Lecture Notes in Computer Science*, (pp. 212–226). Springer.
URL https://doi.org/10.1007/3-540-45635-X_22

Denecker, M., & Vennekens, J. (2008). Building a knowledge base system for an integration of logic programming and classical logic. In M. G. de la Banda, & E. Pontelli (Eds.) *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, vol. 5366 of *Lecture Notes in Computer Science*, (pp. 71–76). Springer.
URL https://doi.org/10.1007/978-3-540-89982-2_12

Denecker, M., & Vennekens, J. (2014). The well-founded semantics is the principle of inductive definition, revisited. In C. Baral, G. D. Giacomo, & T. Eiter (Eds.) *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press.
URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7957

Deryck, M., Vennekens, J., Devriendt, J., & Marynissen, S. (2019). Legislation in the knowledge base paradigm: Interactive decision enactment for registration duties. In *13th IEEE International Conference on Semantic Computing, ICSC 2019, Newport Beach, CA, USA, January 30 - February 1, 2019*, (pp. 174–177). IEEE.
URL https://doi.org/10.1109/ICOSC.2019.8665543

Devriendt, J., Bogaerts, B., Bruynooghe, M., & Denecker, M. (2016). On local domain symmetry for model expansion. *Theory Pract. Log. Program.*, *16*(5-6), 636–652.

Drake, R. G. (2014). The nurse rostering problem: from operational research to organizational reality? *Journal of advanced nursing*, *70*(4), 800–810.

Enderton, H. B. (1972). *A mathematical introduction to logic*. Academic Press.

Falkner, A. A., Haselböck, A., Krames, G., Schenner, G., & Taupe, R. (2019). Constraint solver requirements for interactive configuration. In L. Hotz, M. Aldanondo, & T. Krebs (Eds.) *Proceedings of the 21st Configuration Workshop*, vol. 2467 of *CEUR Workshop Proceedings*, (pp. 65–72).
URL http://ceur-ws.org/Vol-2467/paper-12.pdf

Feinerer, I., Salzer, G., & Sisel, T. (2011). Reducing multiplicities in class diagrams. In J. W. et al. (Ed.) *MODELS 2011.*, vol. 6981 of *Lecture Notes in Computer Science*, (pp. 379–393). Springer.

Felfernig, A., Hotz, L., Bagley, C., & Tiihonen, J. (2014). *Knowledge-based configuration: From research to business cases*. Newnes.

Fitting, M. (1985). A Kripke-Kleene semantics for logic programs. *J. Log. Program.*, *2*(4), 295–312.
URL https://doi.org/10.1016/S0743-1066(85)80005-4

Fitting, M. (2020). Intensional Logic. In E. N. Zalta (Ed.) *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2020 ed.

Friedrich, G., Ryabokon, A., Falkner, A. A., Haselböck, A., Schenner, G., & Schreiner, H. (2011). (Re)configuration using answer set programming. In Kostyantyn M. Shchekotykhin et al. (Ed.) *Proceedings of the IJCAI 2011 Workshop on Configuration*, vol. 755 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In *ICLP/SLP*, vol. 88, (pp. 1070–1080). Cambridge, MA.

Gent, I. P., Petrie, K. E., & Puget, J.-F. (2006). Symmetry in constraint programming. *Foundations of Artificial Intelligence*, *2*, 329–376.

Goossens, A., Vandevelde, S., Vanthienen, J., & Vennekens, J. (2023). GPT-3 for decision logic modeling. In J. Vanthienen, T. Kliegr, P. Fodor, D. Lanti, D. Arndt, E. V. Kostylev, T. Mitsikas, & A. Soylu (Eds.) *Proceedings of the 17th International Rule Challenge and 7th Doctoral Consortium @ RuleML+RR 2023 co-located with 19th Reasoning Web Summer School (RW 2023) and 15th DecisionCAMP 2023 as part of Declarative AI 2023, Oslo, Norway, 18 - 20 September, 2023*, vol. 3485 of *CEUR Workshop Proceedings*. CEUR-WS.org.
URL https://ceur-ws.org/Vol-3485/paper3896.pdf

Guns, T. (2019). Increasing modeling language convenience with a universal n-dimensional array, CPpy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, vol. 19.

Guo, W., Zhen, H.-L., Li, X., Luo, W., Yuan, M., Jin, Y., & Yan, J. (2023). Machine learning methods in solving the boolean satisfiability problem. *Machine Intelligence Research*, *20*(5), 640–655.
URL https://www.mi-research.net/en/article/doi/10.1007/s11633-022-1396-2

Gupta, G., Bansal, A., Min, R., Simon, L., & Mallya, A. (2007). Coinductive logic programming and its applications. In V. Dahl, & I. Niemelä (Eds.) *Logic*

Programming, 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007, Proceedings, vol. 4670 of *Lecture Notes in Computer Science*, (pp. 27–44). Springer.
URL https://doi.org/10.1007/978-3-540-74610-2_4

Hadzic, T., & Andersen, H. R. (2004). An introduction to solving interactive configuration problems. *Techn. Ber. TR-2004-49, The IT University of Copenhagen*.

Hill, P., & Gallagher, J. (1998). Meta-programming in logic programming. *Handbook of Logic in Artificial Intelligence and Logic Programming*, *5*, 421–497.

Ishay, A., Yang, Z., & Lee, J. (2023). Leveraging large language models to generate answer set programs. In P. Marquis, T. C. Son, & G. Kern-Isberner (Eds.) *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, (pp. 374–383).
URL https://doi.org/10.24963/kr.2023/37

Janhunen, T. (2004). Representing normal programs with clauses. In *ECAI*, vol. 16, (p. 358).

Janota, M., Lynce, I., & Marques-Silva, J. (2015). Algorithms for computing backbones of propositional formulae. *Ai Communications*, *28*(2), 161–177.

Jansen, J., Bogaerts, B., Devriendt, J., Janssens, G., & Denecker, M. (2016). Relevance for SAT(ID). In S. Kambhampati (Ed.) *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, (pp. 596–602). IJCAI/AAAI Press.
URL http://www.ijcai.org/Abstract/16/091

Jordens, J., Vandevelde, S., Van Doninck, B., Witters, M., & Vennekens, J. (2022). Adhesive selection via an interactive, user-friendly system based on symbolic AI. *Procedia CIRP*.

Kuncak, V., & Rinard, M. C. (2007). Towards efficient satisfiability checking for boolean algebra with Presburger arithmetic. In F. Pfenning (Ed.) *CADE-21, 2007*, vol. 4603 of *Lecture Notes in Computer Science*, (pp. 215–230). Springer.

Lifschitz, V. (2008). What is answer set programming? In D. Fox, & C. P. Gomes (Eds.) *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, (pp. 1594–1597). AAAI Press.
URL http://www.aaai.org/Library/AAAI/2008/aaai08-270.php

Liu, K., Löffler, S., & Hofstedt, P. (2019). Social golfer problem revisited. In H. J. van den Herik, A. P. Rocha, & L. Steels (Eds.) *Agents and Artificial Intelligence - 11th International Conference, ICAART 2019, Prague, Czech Republic, February 19-21, 2019, Revised Selected Papers*, vol. 11978 of *Lecture Notes in Computer Science*, (pp. 72–99). Springer.
URL https://doi.org/10.1007/978-3-030-37494-5_5

Liu, Y. A., & Stoller, S. D. (2020). Founded semantics and constraint semantics of logic rules. *Journal of Logic and Computation*, *30*(8), 1609–1668.

Liu, Y. A., & Stoller, S. D. (2022). Recursive rules with aggregation: a simple unified semantics. *Journal of Logic and Computation*, *32*(8), 1659–1693.
URL https://doi.org/10.1093/logcom/exac072

Lynce, I., & Marques-Silva, J. (2004). On computing minimum unsatisfiable cores. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*.
URL http://www.satisfiability.org/SAT04/programme/110.pdf

Madsen, J. N. (2003). *Methods for interactive constraint satisfaction*. Master's thesis, Department of Computer Science, University of Copenhagen.

Markovic, D., Bruynooghe, M., & Denecker, M. (2023). Towards systematic treatment of partial functions in knowledge representation.

Marques-Silva, J. P., & Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

Marynissen, S., Bogaerts, B., & Denecker, M. (2021). On the relation between approximation fixpoint theory and justification theory. In Z. Zhou (Ed.) *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, (pp. 1973–1980). ijcai.org.
URL https://doi.org/10.24963/ijcai.2021/272

Marynissen, S., Heyninck, J., Bogaerts, B., & Denecker, M. (2022). On nested justification systems. *Theory and Practice of Logic Programming*, *22*(5), 641–657.

McCarthy, J. (1989). Artificial intelligence, logic and formalizing common sense. In *Philosophical logic and artificial intelligence*, (pp. 161–190). Springer.

McCarthy, J. (1998). Elaboration tolerance. In *Common Sense*, vol. 98. Citeseer.

Meng, B., Reynolds, A., Tinelli, C., & Barrett, C. W. (2017). Relational constraint solving in SMT. In L. de Moura (Ed.) *CADE 26, 2017*, vol. 10395 of *Lecture Notes in Computer Science*, (pp. 148–165). Springer.

Mitchell, D. G., Ternovska, E., Hach, F., & Mohebali, R. (2006). Model expansion as a framework for modelling and solving search problems. Tech. Rep. TR 2006-24, Simon Fraser University, Canada.

Mittal, S., & Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. In H. E. Shrobe, T. G. Dietterich, & W. R. Swartout (Eds.) *Proceedings of the 8th National Conference on Artificial Intelligence*, (pp. 25–32). AAAI Press / The MIT Press.
URL http://www.aaai.org/Library/AAAI/1990/aaai90-004.php

Nute, D. (2001). Defeasible logic. In *Proceedings of the 14th International Conference on Applications of Prolog, INAP 2001, University of Tokyo, Tokyo, Japan, October 20-22, 2001*, (pp. 87–114). The Prolog Association of Japan.

Pelov, N., & Ternovska, E. (2005). Reducing inductive definitions to propositional satisfiability. In *International Conference on Logic Programming*, (pp. 221–234). Springer.

Piller, F. T., Harzer, T., Ihl, C., & Salvador, F. (2014). Strategic capabilities of mass customization based E-Commerce: Construct development and empirical test. In *47th Hawaii International Conference on System Sciences, HICSS 2014, Waikoloa, HI, USA, January 6-9, 2014*, (pp. 3255–3264). IEEE Computer Society.
URL https://doi.org/10.1109/HICSS.2014.403

Randall, T., Terwiesch, C., & Ulrich, K. T. (2005). Principles for user design of customized products. *California Management Review*, *47*(4), 68–85.

Reger, G., Suda, M., & Voronkov, A. (2016). Finding finite models in multi-sorted first-order logic. In Nadia Creignou et al. (Ed.) *Theory and Applications of Satisfiability Testing - SAT 2016*, vol. 9710 of *Lecture Notes in Computer Science*, (pp. 323–341). Springer.

Reiter, R. (1980). A logic for default reasoning. *Artif. Intell.*, *13*(1-2), 81–132.
URL https://doi.org/10.1016/0004-3702(80)90014-4

Schenner, G., & Taupe, R. (2017). Techniques for solving large-scale product configuration problems with ASP. In *Proceedings of the 19th International Configuration Workshop*, (pp. 12–19).

Schneeweiss, D., & Hofstedt, P. (2011). FdConfig: a constraint-based interactive product configurator. In *International Conference on Applications*

of Declarative Programming and Knowledge Management, (pp. 239–255). Springer.

Smith, B. M. (2001). Reducing symmetry in a combinatorial design problem. *Research Report Series-University of Leeds School of Computer Studies LU SCS RR*, (1).

Stumptner, M., & Haselböck, A. (1993). A generative constraint formalism for configuration problems. In P. Torasso (Ed.) *Proceedings of Advances in Artificial Intelligence*, vol. 728 of *Lecture Notes in Computer Science*, (pp. 302–313). Springer.
URL https://doi.org/10.1007/3-540-57292-9_68

Suster, S., Fivez, P., Totis, P., Kimmig, A., Davis, J., De Raedt, L., & Daelemans, W. (2021). Mapping probability word problems to executable representations. In M. Moens, X. Huang, L. Specia, & S. W. Yih (Eds.) *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, (pp. 3627–3640). Association for Computational Linguistics.
URL https://doi.org/10.18653/v1/2021.emnlp-main.294

Suter, P., Steiger, R., & Kuncak, V. (2011). Sets with cardinality constraints in satisfiability modulo theories. In R. J. et al. (Ed.) *VMCAI 2011*, vol. 6538 of *Lecture Notes in Computer Science*, (pp. 403–418). Springer.

Taupe, R., Falkner, A., & Schenner, G. (2016). Deriving tighter component cardinality bounds for product configuration. In *18 th International Configuration Workshop*, (p. 47).

Textor, M. (2021). States of Affairs. In E. N. Zalta (Ed.) *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2021 ed.

Torlak, E., & Jackson, D. (2007). Kodkod: A relational model finder. In O. Grumberg, & M. Huth (Eds.) *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, vol. 4424 of *Lecture Notes in Computer Science*, (pp. 632–647). Springer.
URL https://doi.org/10.1007/978-3-540-71209-1_49

van der Hallen, M., & Janssens, G. (2018). SOGrounder: Modelling and solving second-order logic. In *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*.

Van Hertum, P., Dasseville, I., Janssens, G., & Denecker, M. (2017). The KB paradigm and its application to interactive configuration. *Theory Pract. Log. Program.*, *17*(1), 91–117.
URL https://doi.org/10.1017/S1471068416000156

Vandevelde, S., Jordens, J., Van Doninck, B., Witters, M., & Vennekens, J. (2022). Knowledge-based support for adhesive selection. In G. Gottlob, D. Inclezan, & M. Maratea (Eds.) *Logic Programming and Nonmonotonic Reasoning*, (pp. 445–455). Cham: Springer International Publishing.

Vennekens, J., Denecker, M., & Bruynooghe, M. (2009). CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and practice of logic programming*, *9*(3), 245–308.

Vlaeminck, H., Vennekens, J., Denecker, M., & Bruynooghe, M. (2012). An approximative inference method for solving $\exists\forall so$ satisfiability problems. *Journal of Artificial Intelligence Research*, *45*, 79–124.

Wang, H. (1952). Logic of many-sorted theories. *The Journal of Symbolic Logic*, *17*(2), 105–116.

Zhang, Y., Pu, G., & Zhang, M. (2019). SMTBCF: efficient backbone computing for SMT formulas. In Y. A. Ameur, & S. Qin (Eds.) *Formal Methods and Software Engineering - 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5-9, 2019, Proceedings*, vol. 11852 of *Lecture Notes in Computer Science*, (pp. 36–51). Springer.
URL https://doi.org/10.1007/978-3-030-32409-4_3

Alphabetical Index

abstraction relation, 13
accessor, 18
aggregate, 19, 26
answer set programming, 15
approximation fixpoint theory, 7
argument, 4
arity, 4
atom, 4
AutoConfig, 39
automorphism, 69

backbone, 70
binary quantification, 26
body, 6
Boolean Algebra with Pressburger Arithmetic, 113

Clark's completion, 7
Closed World Assumption, 18, 56
company control example, 9
concept, 22
configuration problem
    generative, 99
constant, 4
constructor, 18
cycle in a permutation, 68

definition, 6, 19, 27
    (non-)monotone, 8
    inductive, 6
    parameter, 6
    semantics, 7, 15, 19, 29
Domain Closure Axiom, 18
domain element, 5
domain of discourse, 5

elaboration tolerance, 23
equisatisfiability, 82
expansion
    regular lifted structure, 74
    variable assignment, 82
extension (of symbol), 22

fact, 4
FO(ID), 6
FO-dot, 15
formula, 4
function
    predefined, 76
    regular, 74

graph of function, 9
ground expression, 4

head, 6

identifier, 18
IDP-Z3, 24
IDP3, 39
induction order, 9
inference
    explanation, 51

propagation, 44, 50
relevance, 47, 51, 52
informal semantics, 12
intension (of symbol), 22
Interactive Consultant, 49
interpretation, 5

justification theory, 7
nested, 10

knowledge base, 25
knowledge base paradigm, 37
knowledge representation, 11
vs. programming, 15, 29, 116

laws of acceptable solutions, 36, 40
laws of possible environments, 36, 40
logic
default, 10
defeasible, 10
first-order, 4
intensional, 22
modal, 23
second-order, 33
logic connective, 4
logic program, 7

meta-programming, 23
model, 6
model expansion, 14
interactive, 41
lifted, 65
split, 40
temporal, 54
model optimization, 14

observation, 36, 43
operator
modal, 23
rule, 6
orbit, 68

partial function, 19, 31
permutation, 68
pigeonhole problem, 65, 71–73, 81, 96
predicate
predefined, 76
problem class, 14, 116
problem domain, 14
programming vs. KR, 15, 29, 116
propagation, 44, 50

quantification, 4

Rack configuration problem, 3
Railway Network problem, 2
reachability relation, 6, 29
Real Estate Sales Tax problem, 2
regularity, 74
function, 74, 76, 77
lifted structure, 74
lifted tuple, 75
reification, 23
relevance, 47, 51, 52
rule, 6

SAT solver, 16
satisfaction, 6
satisfiability checking, 6
second-order logic, 22
sentence, 4
SMT solver, 16
solution
contingent, 46
definite, 45, 48, 49
incomplete, 47–49
state of affairs, 1
stratification, 8
structure, 5
lifted, 72
partial, 5
precision, 5
regular, lifted, 74
total, 5
symbol

decision, 36, 40
environmental, 36, 40
function, 4
nullary, 4
predicate, 4
unary, 4
symmetry breaking, 68

term, 4
theory, 4
transitive closure, 29
type signature, 17

uncertainty, 14, 37
Unique Name Axiom, 18
universally solvable, 15, 31, 46

variable, 4
free, 4
variable assignment, 5
vocabulary, 4
temporal, 55



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
pierre.carbonnelle@kuleuven.be