Simulating Dynamic Systems Using Linear Time Calculus Theories

Bart Bogaerts, Joachim Jansen, Maurice Bruynooghe,

Broes De Cat, Joost Vennekens, and Marc Denecker

Department of Computer Science, KU Leuven

(*e-mail:* {firstname.lastname}@cs.kuleuven.be)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Dynamic systems play a central role in fields such as planning, verification, and databases. Fragmented throughout these fields, we find a multitude of languages to formally specify dynamic systems and a multitude of systems to reason on such specifications. Often, such systems are bound to one specific language and one specific inference task. It is troublesome that performing several inference tasks on the same knowledge requires translations of your specification to other languages. In this paper we study whether it is possible to perform a broad set of well-studied inference tasks on one specification. More concretely, we extend IDP^3 with several inferences from fields concerned with dynamic specifications.

KEYWORDS: Dynamic systems, progression, simulation, knowledge base system, inferences.

1 Introduction

Traditionally, systems that reason on declarative specifications take input in a specific language and perform one specific inference on this input. As argued in Denecker and Vennekens (2008) and Denecker (2012), it is often useful to perform several different inference tasks on the same knowledge base; the authors called this idea the Knowledge Base System (KBS) paradigm. In this paper, we evaluate the usefulness of the KBS paradigm in the well-studied domain of dynamic action languages. We identify many interesting inference tasks in this domain and we show that for one concrete action language, the Linear Time Calculus (LTC) introduced in this paper, each of these tasks can be performed. As a result, we can use the same specification, an LTC-theory, for performing a wide range of tasks, whereas traditional software development uses different specifications for different tasks. We illustrate this with different tasks related to development of a Pac-Man game.

We do not start from scratch, our general approach reduces inference tasks for LTC-theories to existing inference methods. We do this in the context of the IDP^3 system (De Cat et al. 2014), a KBS that allows manipulation of logical objects (theories, structures, terms, queries,. . .) through an imperative layer and hence allows users to glue the different inference methods together to construct useful software (De Pooter et al. 2011). For IDP^3, the imperative layer is the Lua scripting language (Ierusalimschy et al. 1996).

Many action calculi have been developed, including the Situation Calculus (McCarthy and Hayes 1969; Reiter 2001), Event Calculus (Kowalski and Sergot 1986), and the Planning Domain Definition Language (PDDL) (Ghallab et al. 1998) and many inference methods exist for these calculi. *Progression inference* aims at finding the successor of a given state, and plays a central role in database applications (Lin and Reiter 1997; Kowalski and Sadri 2013). *Simulation* uses progression to simulate the execution of a system based on a formal specification. *Planning* aims at finding a sequence of actions that achieve a goal. Other inference tasks are finding and/or proving *invariants* of a dynamic system, and *verifying* complex temporal statements such as LTL or CTL expressions.

Even though many important inference methods—including theorem proving (Fitting 1996), (optimal) model expansion (Wittocx et al. 2008), querying (Vardi 1986) and debugging (Shlyakhter et al. 2003)—are already supported by IDP^3, several inferences in a dynamic context, such as progression, simulation, and proving invariants, are not yet supported. To overcome this limitation, we show that all of the above inferences can be implemented in a subclass of theories, the so-called Linear Time Calculus (LTC) theories.

From a dynamic (time-dependent) LTC-theory, we derive two simpler static theories—an initial theory and a transition theory—and we show that progression on the LTC-theory can be performed by model expansion on the transition theory and that simulation can be achieved by repeated progression. Proving invariants is achieved by induction: by proving it for the initial theory and proving that the property is preserved by the transition theory. Finally we discuss how one could handle more complex dynamic properties.

The main contributions of this paper are threefold: i) we illustrate the practical advantages of the KBS paradigm in the context of dynamic systems, ii) we implement methods to perform progression and simulation and to prove invariants on the same theory, and iii) we study the relation between various declarative problem-solving domains concerned with dynamic systems and identify which inferences are studied in which domains.

The paper is organised as follows. In Section 2 we recall some preliminaries. Next, in Section 3, we introduce the class of structures and theories of interest: structures describing an evolution of a state over time and theories that essentially contain only local information. Afterwards, in Section 4, we provide an overview of the inferences applicable to these theories and in Section 5 we compare with other systems. We conclude in Section 6. Omitted proofs can be found in Appendix B.

2 Preliminaries

We assume familiarity with basic concepts of first-order logic (FO). If J is a structure over Σ, and σ a symbol in vocabulary Σ, σ^J denotes the interpretation of σ in J. If φ is a formula and t_1 and t_2 are terms, we use $\varphi[t_2/t_1]$ for the formula obtained from φ by replacing all occurrences of t_1 by t_2. We use a many-typed logic and write $P(t_1,\ldots,t_n)$ and $f(t_1,\ldots,t_n) : t'$ for the predicate P typed $t_1,\ldots,t_n$ respectively the function f with input arguments of type $t_1,\ldots,t_n$ and output argument typed t'.

FO(ID) extends FO with (inductive) definitions: sets of rules of the form $\forall\overline{x} : P(\overline{t}) \leftarrow \varphi$, (or $\forall\overline{x} : f(\overline{t}) = t' \leftarrow \varphi$) where φ is a FO formula and the free variables of φ and $P(\overline{t})$ are among the $\overline{x}$. We call $P(\overline{t})$ (respectively $f(\overline{t}) = t'$) the head of the rule and φ the body. The connective $\leftarrow$ is the *definitional implication*, which should not be confused with the material implication $\Rightarrow$. Thus, the expression $\forall\overline{x} : P(\overline{t}) \leftarrow \varphi$ is *not* a shorthand for

$\forall \bar{x} : P(\bar{t}) \lor \neg\varphi$. Instead, its meaning is given by the well-founded semantics (for functions, semantics of the graph predicate is considered, i.e., as if the rule were $Graph_f(\bar{t}, t) \leftarrow \varphi$); this semantics correctly formalises all kinds of definitions that typically occur in mathematical texts (Denecker and Ternovska 2008; Denecker and Vennekens 2014).

To simplify the presentation, we assume, without loss of generality, that an FO(ID) theory consists of a set of FO sentences and a single definition (Denecker and Ternovska 2008; Mariën et al. 2004); our implementation does not impose this restriction.

3 Linear Time Calculus

We define linear-time vocabularies and structures. Next, we define the progression inference and analyse when progression can be performed without keeping an explicit history (when a theory satisfies the Markov property (Markov 1906)). Finally, we define the Linear Time Calculus, and show that LTC-theories satisfy the Markov property.

3.1 Linear-Time Vocabularies and Structures

Definition 3.1 (*Linear-time vocabulary*)
A *linear-time vocabulary* is a many-typed first-order vocabulary Σ such that:

- Σ has a type $Time$ (always interpreted as $\mathbb{N}$), a constant $\mathcal{I}$ of type $Time$ (interpreted as 0) and a function $\mathcal{S}(Time) : Time$ (interpreted as the successor function),
- All other symbols in Σ have at most one argument of type $Time$,
- Apart from $\mathcal{I}$ and $\mathcal{S}$, the output argument of functions is not of type $Time$.

We partition symbols in Σ in three categories: $Time$, $\mathcal{I}$ and $\mathcal{S}$ are LTC-symbols, symbols without a $Time$ argument are *static* symbols, and all other symbols are *dynamic* symbols. For ease of notation, we will assume that the $Time$ always occurs last in dynamic symbols.

In the rest of this paper we assume that Σ is a linear-time vocabulary. Such a structure describes an evolution of a state over time, i.e., it represents a sequence of states. Here, a state is a structure over a vocabulary derived from Σ by projecting out $Time$.

Definition 3.2 (*Projected symbol*)
If $\sigma(t_1, \ldots, t_{n-1}, Time)$ is a dynamic predicate symbol, then $\sigma_{curr}(t_1, \ldots, t_{n-1})$ is its *projected symbol*. Similarly, for a function symbol $\sigma(t_1, \ldots, t_{n-1}, Time) : t$, its *projected symbol* is $\sigma_{curr}(t_1, \ldots, t_{n-1}) : t$.

Definition 3.3 (*Derived vocabularies*)
The vocabularies derived from Σ are:

- the *static vocabulary* Σ_s consisting of all static symbols in Σ;
- the *single-state vocabulary* Σ_{ss} which extends the static vocabulary Σ_s with the symbol σ_{curr} for each dynamic symbol σ in Σ.

Intuitively, a Σ_{ss}-structure describes a single state, and σ_{curr} describes the interpretation of σ on that point in time.

Example 3.4
A simplification of the Pac-Man game can be modelled with a linear-time vocabulary Σ^p consisting of types $Time$, $Agent$, $Square$, and Dir. The vocabulary contains predicate symbols $Next(Square, Dir, Square)$, $Move(Agent, Dir, Time)$, $Pell(Square, Time)$, and $GameOver(Time)$. An atom $Next(s, d, s')$ expresses that the square next to s in direction d is s', $Move(a, d, t)$ that agent a moves in direction d at time t, $Pell(s, t)$ that square s contains a pellet at time t, and $GameOver(t)$ that either all pellets are eaten or Pac-Man is dead at t. The vocabulary also contains a constant *pacman* of type *Agent* and function symbols $Pos(Agent, Time) : Square$ mapping each agent at every time-point to his position, and $StartPos(Agent) : Square$ mapping agents to their initial position.

Σ^p_s and Σ^p_{ss} have the same types as Σ^p. Σ^p_s contains the constant *pacman*, the predicate symbol $Next(Square, Dir, Square)$, and the function $StartPos(Agent) : Square$. Σ^p_{ss} is an extension of Σ^p_s with predicate symbols $Move_{curr}(Agent, Dir)$, $Pell_{curr}(Square)$, and $GameOver_{curr}$ and the function symbol $Pos_{curr}(Agent) : Square$.

Definition 3.5 (*Projection of a structure*)
Let Ω be the interpretation of a dynamic symbol σ/n in a structure and $k \in \mathbb{N}$. The set of tuples $(d_1, \dots, d_{n-1})$ such that $(d_1, \dots, d_{n-1}, k) \in \Omega$ is the *k-projection* of Ω, denoted $\pi_k(\Omega)$.

The *single-state projection* of a Σ-structure J on time $k \in \mathbb{N}$, denoted $\pi^{ss}_k(J)$, is the Σ_{ss}-structure interpreting static symbols σ as σ^J and dynamic symbols σ_{curr} as $\pi_k(\sigma^J)$.

Proposition 3.6
Let Σ be a linear-time vocabulary and Σ_{ss} the corresponding single state vocabulary. There is a one-to-one correspondence between Σ-structures J and sequences $(J_k)_{k=0}^{\infty}$ of Σ_{ss}-structures sharing the same interpretation of static symbols, given by $J_k = \pi^{ss}_k(J)$.

This proposition holds because one can reconstruct the tuples in Ω from the tuples in $\pi_k(\Omega)$. From now on, we will often identify a structure with the corresponding sequence. Often, we are not only interested in single states, but also in two successive states. In the following definition, a Σ_{bs}-structure describes two subsequent states; σ_{curr} refers to the first one and σ_{next} to the second.

Definition 3.7 (*Bistate vocabulary and structure*)
For every dynamic symbol σ in Σ, the *next-state symbol* is a new symbol σ_{next} with the same type signature as σ_{curr}. The *bistate vocabulary* Σ_{bs} extends the single-state vocabulary Σ_{ss} with the symbol σ_{next} for each dynamic symbol. With J a Σ-structure, the *bistate projection* $\pi^{bs}_k(J)$ over Σ_{bs} interprets σ_{next} as $\pi_{k+1}(\sigma^J)$ and all other symbols as in $\pi^{ss}_k(J)$. If S and S' are Σ_{ss}-structures that are equal on static symbols, we use (S, S') for the Σ_{bs}-structure with the same interpretation of static symbols and such that $\sigma^{(S,S')}_{curr} = \sigma^S_{curr}$ and $\sigma^{(S,S')}_{next} = \sigma^{S'}_{curr}$ for dynamic symbols σ.

3.2 Progression and the Markov Property

Definition 3.8 (*k-chain, extension*)
Let k be a natural number. A *k-chain* J over Σ is a sequence $(J_i)_{i=0}^{k}$ of Σ_{ss}-structures sharing the same interpretation of static symbols. Slightly abusing notation, we also call a Σ-structure an ∞-chain (using the identification in Proposition 3.6).

The *direct extension* of a k-chain J with a Σ_{ss} structure S, denoted $J{::}S$, is the $(k+1)$-chain J' with $J'_j = J_j$ for $j \leq k$ and $J'_{k+1} = S$. A chain J'' is an extension of J if it is either a direct extension of J or an extension of a direct extension of J.

Definition 3.9 ($\mathcal{T}$*-compatible,* $\mathcal{T}$*-successor*)
Let $\mathcal{T}$ be a Σ-theory. A k-chain J is $\mathcal{T}$*-compatible* if $\mathcal{T}$ has a model (an ω-chain) that extends J. A Σ_{ss}-structure S' is a $\mathcal{T}$*-successor* of a k-chain J if $J{::}S$ is $\mathcal{T}$-compatible.

Definition 3.10 (*Progression inference*)
Progression inference is an inference that takes as input a theory $\mathcal{T}$ and a $\mathcal{T}$-compatible k-chain J and returns all $\mathcal{T}$-successors of J.

Of special interest is the case where the $\mathcal{T}$-successors of a k-chain J are determined solely by the last state in J. It means that the dynamic system has no history. From a practical point of view this is often important. For example, contemporary databases are often too large to keep track of the entire history. We refer to such a system as a system that has the Markov property (Markov 1906).[1]

Definition 3.11 (*Markov property*)
A theory $\mathcal{T}$ satisfies the *Markov property* if for every $\mathcal{T}$-compatible k-chain J, and every $\mathcal{T}$-compatible k'-chain J' ending in the same state, i.e., such that $J_k = J'_{k'}$, the $\mathcal{T}$-successors of J are exactly the $\mathcal{T}$-successors of J'.

The condition on a k-chain J to be $\mathcal{T}$-compatible is quite strong. It does not only require that all information in this chain is correct according to $\mathcal{T}$, but it requires that J is extensible to a model of $\mathcal{T}$. This might require to look into the future. For example in the Pac-Man game, we could add two constraints: i) agents can not turn back and ii) as the game is not over, every agent moves. These two sentences are contradictory when an agent arrives at the end of a dead-end corridor. This means that every k-chain in which an agent *enters* a dead-end corridor is not $\mathcal{T}$-compatible, as the agent will eventually reach the point where it cannot move. On the one hand, this is a good property, because progression as defined above guarantees that you can never get stuck, that every $\mathcal{T}$-compatible chain can always be progressed. But on the other hand, from a computational point of view, this is bad, as progression requires to look arbitrarily far into the future. In Appendix A, we present the notions of *weak* $\mathcal{T}$*-compatibility*, *weak progression* and the *weak Markov property* which are more technical than the one we described here. Intuitively, these properties are similar to the ones described here, except they do not require looking into the future. This might results in *deadlocks*: chains without successors. We implemented the weak progression; to the best of our knowledge, all systems that implement progression actually implement weak progression.

[1] The Markov property is often used in a probabilistic context. Translated to that context, one could say that the $\mathcal{T}$-successors of J are the states with non-zero probability.

3.3 The Linear Time Calculus

Definition 3.12 (*static, single-state, bistate*)
Let φ be either a sentence or a rule. We call φ *static* if it contains no terms of type $Time$. We call φ *initial* if it contains the constant $\mathcal{I}$ and no other terms of type $Time$. We call φ *single-state* if it contains a variable of type $Time$ and no other terms of type $Time$. We call φ *bistate* if it contains a variable t typed $Time$ and all terms typed $Time$ in φ are either t or $\mathcal{S}(t)$. Furthermore, we call a single-state or a bistate φ *universal* if it is of the form $\forall t : \varphi'$ where t is the unique $Time$-variable in φ and t is not quantified in φ'.

We now define an LTC theory as a theory that roughly only consists of the above types of rules and formulas. In this definition, we use the notion of stratification over $Time$: a definition is stratified over $Time$ if it does not contain any rules defining atoms in terms of future values.

Definition 3.13 (*LTC-theory*)
An *LTC-theory* over Σ is a theory $\mathcal{T}$ that satisfies i) all sentences and rules in $\mathcal{T}$ are either static, initial, universal single-state, or universal bistate, and ii) the definition in $\mathcal{T}$ is stratified over $Time$.

The first condition ensures that an LTC-theory has no history. For pure FO theories, this is enough to guarantee the Markov-property. The second condition prevents nonsensical definitions such as for example defining the state in terms of a future state.

Example 3.14
Below is an LTC theory over Σ^p (Example 3.4) specifying part of the Pac-Man game[2].

$$\left\{\begin{array}{l}
\forall a, p : Pos(a, \mathcal{I}) = p \leftarrow StartPos(a) = p.\\
\forall a, t, p : Pos(a, \mathcal{S}(t)) = p \leftarrow Pos(a, t) = p \land \neg \exists d : Move(a, d, t).\\
\forall a, t, p : Pos(a, \mathcal{S}(t)) = p \leftarrow \exists d : Move(a, d, t) \land Next(Pos(a, t), d, p).\\
\forall s : Pell(s, \mathcal{I}).\\
\forall s, t : Pell(s, \mathcal{S}(t)) \leftarrow Pell(s, t) \land Pos(pacman, t) \neq s.
\end{array}\right\}$$
$$\forall a, t, d, d' : Move(a, d, t) \land Move(a, d', t) \Rightarrow d = d'.$$

The theory inductively defines the positions of the agent and the pellets at each time point (in terms of the open predicates $Next$, $StartPos$ and $Move$) and states the constraint that there is only one move at a time.

We now show how an LTC-theory can be translated automatically into two simpler theories: a Σ_{ss}-theory that describes valid initial states, and a Σ_{bs}-theory that describes valid transitions.

Definition 3.15 (*Elimination of time*)
Let φ be a universal single-state or bistate sentence or rule with unique $Time$ variable t. The *time-elimination* of φ is the sentence/rule $te(\varphi)$ obtained from φ by (i) dropping the universal quantification of t, (ii) for every occurrence of $\mathcal{S}(t)$ in a dynamic (predicate or function) symbol σ, replacing σ by σ_{next} and dropping the argument $\mathcal{S}(t)$, and (iii) for every occurrence of t in a dynamic symbol, replacing σ by σ_{curr} and dropping t.

[2] The complete example can be found at Bogaerts (2014).

For example, the time-elimination of the rule

$$\forall a, t, p : Pos(a, \mathcal{S}(t)) = p \leftarrow \exists d : Move(a, d, t) \land Next(Pos(a, t), d, p).$$

is the Σ_{bs}-rule

$$\forall a, p : Pos_{next}(a) = p \leftarrow \exists d : Move_{curr}(a, d) \land Next(Pos_{curr}(a), d, p).$$

Definition 3.16 (*Initial and transition theory*)
Let $\mathcal{T}$ be an LTC-theory. We define two theories. The *initial theory* $\mathcal{T}_0$ consists of:

- all static sentence/rules in $\mathcal{T}$,
- for each initial sentence/rule φ in $\mathcal{T}$, the sentence/rule $te(\forall t : \varphi[t/\mathcal{I}])$ (informally, here we replace $\mathcal{I}$ by t and project on t afterwards since te is only defined for universal sentences; this is the same as projecting on $\mathcal{I}$),
- for each single-state sentence/rule φ in $\mathcal{T}$, the sentence/rule $te(\varphi)$.

The *transition theory* $\mathcal{T}_t$ consists of:

- all static sentences/rules in $\mathcal{T}$,
- for each single-state sentence/rule φ in $\mathcal{T}$, the sentences/rules $te(\varphi)$ and $te(\varphi[\mathcal{S}(t)/t])$,
- for each bistate sentence/rule φ in $\mathcal{T}$, the sentence/rule $te(\varphi)$.

Example 3.17
For our Pac-Man example, this results in the initial theory $\mathcal{T}_0$:

$$\begin{array}{c}\left\{\begin{array}{l}\forall a, p : Pos_{curr}(a) = p \leftarrow StartPos(a) = p. \\ \forall s : Pell_{curr}(s).\end{array}\right\} \\ \forall a, d, d' : Move_{curr}(a, d) \land Move_{curr}(a, d') \Rightarrow d = d'.\end{array}$$

and the transition theory $\mathcal{T}_t$:

$$\begin{array}{l}\left\{\begin{array}{l}\forall a, p : Pos_{next}(a) = p \leftarrow Pos_{curr}(a) = p \land \neg\exists d : Move_{curr}(a, d). \\ \forall a, t, p : Pos_{next}(a) = p \leftarrow \exists p', d : Pos_{curr}(a) = p' \land Move_{curr}(a, d) \land Next(p', d, p). \\ \forall s, t : Pell_{next}(s) \leftarrow Pell_{curr}(s) \land \neg Pos_{curr}(pacman) = s.\end{array}\right\} \\ \forall a, d, d' : Move_{curr}(a, d) \land Move_{curr}(a, d') \Rightarrow d = d'. \\ \forall a, d, d' : Move_{next}(a, d) \land Move_{next}(a, d') \Rightarrow d = d'.\end{array}$$

We now formalise the relation between $\mathcal{T}$, $\mathcal{T}_0$, and $\mathcal{T}_t$; proofs can be found in Appendix B.

Theorem 3.18
Let $\mathcal{T}$ be an LTC-theory and J a Σ-structure. Then J is a model of $\mathcal{T}$ if and only if $\pi_0^{ss}(J) \models \mathcal{T}_0$ and for every $k \in \mathbb{N}$, $\pi_k^{bs}(J) \models \mathcal{T}_t$.

Theorem 3.19
Let $\mathcal{T}$ be an LTC-theory and J a k-chain. Then, J is weakly $\mathcal{T}$-compatible if and only if $\pi_0^{ss}(J) \models \mathcal{T}_0$ and for every $j < k$, $\pi_j^{bs}(J) \models \mathcal{T}_t$.

Corollary 3.20
LTC-theories satisfy the Markov property and the weak Markov property.

3.4 Modelling Methodology: Actions, Fluents and Inertia

In many action languages, one often divides dynamic predicates into two sets of predicates: *action predicates* and *fluents*. Ever since the frame problem was defined by McCarthy and Hayes (1969), it has been clear that it is often easier to express state changes than it is to express the complete next state. Furthermore, from a practical standpoint, it is often easier to update a state—for example, a database—than it is to compute the entire next state. Therefore, when modelling in LTC we often introduce three extra predicate symbols for each fluent P/n: C_P/n, expresses that P is caused to be true at a certain time, $C_{\neg P}/n$ expresses that P is caused to be false and $I_P/(n-1)$ expresses that P holds initially. The relation between these predicates is formalised in LTC:

$$\left\{\begin{array}{l} \forall \overline{x} : P(\overline{x},\mathcal{I}) \qquad \leftarrow I_P(\overline{x}). \\ \forall \overline{x}, t : P(\overline{x},\mathcal{S}(t)) \leftarrow C_P(\overline{x},t). \\ \forall \overline{x}, t : P(\overline{x},\mathcal{S}(t)) \leftarrow P(\overline{x},t) \land \neg C_{\neg P}(\overline{x},t). \end{array}\right\}$$

I.e., $P(\overline{x},\mathcal{S}(t))$ holds if it is either caused to be true or it was already true and is not caused to be false (inertia). Using this methodology, the modeller simply describes the effects of actions through C_P and $C_{\neg P}$ and a reasoning engine can exploit these new predicates for efficiently updating a persistent state.

4 LTC-Theories in Practice: Inferences and Implementation

This section describes various inference methods we can use on LTC-theories with a brief description of their implementation in IDP^3, available in version 3.3 (IDP 2013).

4.1 Progression

Based on Theorem 3.19, we can use model expansion on the initial theory to infer an initial state and model expansion on the transition theory to infer a next state from a given state. To perform these inferences, we added two procedures to IDP^3:

- `initialise(T,J)` takes as input an LTC-theory T over Σ and a partial Σ-structure J (a structure that at least interprets all types) and returns a set of Σ_{ss}-structures that are initial states of T and that agree with J (that expand $\pi_0^{ss}(J)$). The number of generated Σ_{ss}-structures depends on the option `nbmodels`.
- `progress(T,S)` takes as input an LTC-theory T over Σ and any Σ_{ss}-structure S and returns a set of Σ_{ss}-structures S' for which $(S,S') \models T_t$. The number of generated Σ_{ss}-structures depends on the option `nbmodels`.

4.2 Logic-Based Software Development using Interactive Simulation

An LTC-theory describes the evolution of a dynamic system over time. By itself, it cannot interact with the external world. In order to create such software, we can interactively simulate an LTC-theory by waiting for user input at each progression step. The simplest form is a procedure that uses the progression inference to present all possible next states to a user, and asks to pick one. We implemented this form of interactive simulation in IDP^3: calling `simulate_interactive(T,J)`, with a Σ-LTC-theory T and a partial Σ-structure J at least interpreting all types in Σ, provides you with an interactive shell to guide the simulation. Interactive simulation reuses the `initialise` and `progress` procedures. Due to its rather primitive communication with the user, this kind of simulation is not yet useful for running software based on a logical specification. In order to do so, we need a more refined form of interaction.

4.2.1 Modelling Methodology: Exogenous and Endogenous Information

One way to achieve a more refined form is by making an explicit distinction between *exogenous* and *endogenous* information, respectively information determined by the environment and information internal to the system. In most applications, fluents are endogenous and (a subset of the) actions are exogenous. For Pac-Man, the only exogenous information is the action the player takes: the direction in which he moves. In order to allow such a distinction (and in the meantime, many other refined control mechanisms), we implemented `simulate(`T`,` J`, rand, show(), endcheck(), choose())`. Besides an LTC-theory T and a partial structure J, this inference takes as input:

- `rand`: a boolean; if true, simulations happens randomly, otherwise interactively,
- (optional) `show()` a Lua-procedure that implements printing of the current state,
- (optional) `endcheck()` a Lua-procedure that decides whether to stop the simulation,
- (optional) `choose()` a Lua-procedure that implements choosing a next state.

This procedure also simulates T, but all communication goes through the user provided procedures. It reuses the `initialise` and `progress` procedures. We used the above procedure to simulate a game of Pac-Man. As `show` procedure, we passed a call to the visualisation tool ID^{P}_{Draw} (IDPDraw 2012); the stop-criterion checks for the atom "*GameOver*", and our `choose` procedure asks the user which direction to go to. This results in a complete, playable Pac-Man implementation that can be found at Bogaerts (2014). At the moment, behaviour of the ghosts is random, but this could easily be replaced by smart AI by providing a specification for the behaviour of the ghosts.

4.3 Proving Invariants

Definition 4.1
An *invariant* of an LTC-theory $\mathcal{T}$ is a universal single-state sentence φ such that $\mathcal{T} \models \varphi$.

The straightforward way to prove invariants is theorem proving (deduction inference). In IDP^3, this can be done using the procedure `entails(T,f)`, which checks whether sentence f is entailed by theory T. IDP^3 automatically translates this call to a theorem prover supporting TFA (Sutcliffe et al. 2012) or FOF (Sutcliffe 2009). Often, theorem provers are unable to prove entailed invariants. This can happen for example because the nature of $Time$ ($\mathbb{N}$) is not exploited enough or because this problem is undecidable in general. The following theorem shows that for LTC-theories, we can prove invariants by induction.

Theorem 4.2
Let $\mathcal{T}$ be an LTC-theory and φ a universal single-state sentence. Then $\mathcal{T} \models \varphi$ if $\mathcal{T}_0 \models te(\varphi)$, and $(\mathcal{T}_t \wedge te(\varphi)) \models te(\varphi[\mathcal{S}(t)/t])$, where t is the unique time-variable in φ.

We added a procedure `isinvariant(T,f)` to the IDP^3 system; this procedure checks the two entailment relations from Theorem 4.2. It uses the same transformations as the progression inference and reuses the existing deduction inference of IDP^3.

4.3.1 Fixed-Domain Invariants

Some sentences are not invariants in general, but only in a certain context (i.e., in a given domain). For example: in a given Pac-Man grid it might be that Pac-Man can always reach all remaining pellets. This is however not a general invariant of the Pac-Man game since it is possible to construct grids in which some pellets are completely surrounded by walls. In case a finite domain for all other types than $Time$ and an interpretation for some static symbols is given, it suffices to search for counterexamples of the invariant in this specific setting.

Theorem 4.3
Let J be a Σ_s-structure and let φ be a universal single-state sentence with time variable t. Then φ is satisfied in all Σ-structures expanding J if $\mathcal{T}_0 \land \neg te(\varphi)$ has no models expanding J, and $\mathcal{T}_t \land te(\varphi) \land \neg te(\varphi[\mathcal{S}(t)/t])$ has no models expanding J.

We added the procedure `isinvariant(T,f,J)` to IDP^3; this procedure checks whether sentence f is an invariant of T in the context of J using Theorem 4.3. It reuses the transformations implemented for progression and the model expansion inference of IDP^3.

4.3.2 More General Properties

Proving invariants is often useful. But in many cases one is interested in proving more general properties. For example in the Pac-Man game, a desired property would be that pellets never reappear: $\forall t, s : \neg Pell(s,t) \Rightarrow \neg Pell(s, \mathcal{S}(t))$. This sentence is a universal bistate sentence. For these formulas, we find a result similar to Theorem 4.2.

Theorem 4.4
Let $\mathcal{T}$ be an LTC-theory and φ a universal bistate sentence. Then $\mathcal{T} \models \varphi$ iff $\mathcal{T}_t \models te(\varphi)$.

The above theorem not only yields a method to prove bistate invariants, but also a method to prove them in the context of a given domain similar to Theorem 4.3. The procedures `isinvariant(T,f)` and `isinvariant(T,f,J)` automatically detect whether sentence f is a bistate or a single-state invariant and apply the appropriate methods. For proving more complex properties φ, the only method available yet is directly proving that $\mathcal{T} \models \varphi$.

4.4 ***Planning***

For dynamic domains, *planning* is an important computational task: finding a sequence of actions reaching a certain goal state. To do this in IDP^3, one typically creates a second theory describing the goal state. As an example, the condition that Pac-Man wins, $\exists t : \forall s : \neg Pell(s,t)$, is a goal state. A plan can then be searched through model expansion inference, after merging the LTC-theory with the goal theory. In the standard setting, this requires all domains (including $Time$) to be finite, but recent work on Lazy Grounding removes this restriction (De Cat et al. 2012).

Often, a cost is associated with each plan, e.g., the number of steps needed to win the game. The minimisation inference in IDP^3 searches for a plan with minimal cost.

5 Related Work

Many action languages are closely related to LTC; the relation between several of these calculi has been studied intensively by Thielscher (2011). The focus of this paper is not on the language, but on the forms of inference we can perform. In what follows, we discuss several domains and systems concerned with inference for (dynamic) languages. We show that IDP^3 distinguishes itself by the variety of inferences it offers. An overview of the discussed domains and systems can be found in Table 1.

	IDP^3	DS	Pl	TP	ASP-CP	NuSMV	LPS	ProB
Progression	×	×	−	−	−	×	×	×
Planning	×	−	×	−	×	×	×	×
Optimal Planning	×	−	×	−	×	−	−	×
Proving Invariants	×	−	−	×	−	−	−	−
Interactive Simulation	×	×	−	−	−	×	×	×
LTL/CTL Model Checking	−	−	−	−	−	×	−	×

Table 1. *The various inferences (rows) and systems/fields (columns) we consider in this comparison:* IDP^3, *Database Systems (DS), Planners (Pl), Theorem Provers (TP), ASP/CP-solvers (ASP-CP), NuSMV, LPS and ProB.*

Many database systems implement some form of progression (Lin and Reiter 1997). Often, these systems use (a variant of) transaction logic (Bonner et al. 1993) to express progression steps. Other dynamic inferences, such as backwards reasoning, planning, and verification are, to the best of our knowledge, not possible in these systems. A very interesting database system is LogicBlox (Green et al. 2012); it supports a refined interactive simulation by means of a huge set of built-in predicates (windows, buttons, etc.). Users can specify workflows declaratively; during simulations, the UI is derived from the interpretations of the built-ins.

Proving invariants can be handled by theorem provers such as SPASS (Weidenbach et al. 2009), Vampire (Riazanov and Voronkov 2002), and many more. Sutcliffe (2013) desbribed an overview of state-of-the-art theorem provers. Provers are only able to handle one form of inference, namely deduction. We optimised this for the case of proving invariants of an LTC-theory using induction. Some interactive theorem provers, for example ACL2 (Kaufmann et al. 2000) and Coq (The Coq development team 2004), can generate inductive proofs but they require guidance from the user.

Another community with great interest in dynamic specifications is the *planning* community. Many planners support the PDDL language (Ghallab et al. 1998); Amanda et al. (2012) published an overview of such systems. To the best of our knowledge, these systems only support one form of inference, namely planning. The planning inference is in fact a special case of model expansion. This is demonstrated for example by a tool that translates PDDL specifications into LTC-theories (van Ginkel 2013). Planning problems can also be encoded in other systems that essentially perform model expansion, such as Answer Set Programming (ASP) (Gelfond and Lifschitz 1998) systems, or using constraint programming (Apt 2003) or (integer) linear programming (Nemhauser and Wolsey 1988). In ASP, systems that perform other inferences on dynamic systems have been developed as well. For example oClingo (Gebser et al. 2012) allows stream reasoning, a form of interactive simulation and (Haufe et al. 2012) describes methods to prove invariants of (temporal) ASP encodings of game specifications. However, these various methods have not been unified in one system to work on the same specification.

The above discussion focuses on general fields tackling (only) one of the problems we are typically interested in (in a dynamic context). To be fair, it is worth mentioning that systems in these domains often tackle a more general problem (for example ASP systems can do much more than only planning). IDP^3 tackles these more general problems efficiently as well. Over the years, IDP and MiniSAT(ID) (the solver underlying IDP^3) have proven to be among the best ASP and CP systems (Calimeri et al. 2011; Alviano et al. 2013; Amadini et al. 2013).

Many other systems are designed for dynamic domains. For example, NuSMV (Cimatti et al. 2002) supports progression, interactive simulation, CTL and LTL model checking, and planning (by giving counterexamples for the LTL statement that the goal cannot be reached). This system is propositional, hence symbolic, domain-independent proving of invariants is impossible. CTL and LTL model checking are currently not supported by IDP^3. However, conceptually they form no problem: LTL properties can be translated into Σ-sentences and deduction inference could be used to prove them. Furthermore, progression inference could be used to generate a state graph, on which more efficient CTL and LTL and model checking algorithms can be applied. This is not yet implemented.

The LPS framework from Kowalski and Sadri (2013) has a lot of goals in common with our work, it aims at providing a unified framework for computing with dynamic systems. The language is richer than LTC as rules can relate more than two points in time and there is an explicit representation of external events. The model-theoretic semantics is pretty close to the FO(ID) semantics. The operational semantics corresponds to simulation: it works on time-eliminated states and selects a single successor state. Similar to weak progression, it cannot look in the future, and hence might result in a deadlock. The current implementation is on top of Prolog and mainly aims at (interactive) simulation.

The ProB system (Leuschel and Butler 2008) is an automated animator and model checker for the B-Method. It can provide interactive animations (interactive simulation) and can also be used to do (optimal) planning and automatically verify dynamic specifications. ProB is a very general and powerful system. The only inference studied in this paper it does not support is domain independent proving of invariants.

6 Conclusion and Future Work

In this paper we studied how the KBS paradigm can be applied in the context of dynamic domains. We identified many interesting forms of dynamic inference and explained how all of these inferences can be applied in the context of software development based on logic. We showed that in principle, each of these inferences can be applied on the same problem specification, and thus argued the importance of knowledge reuse.

Furthermore, we implemented, with relative ease, all but one of these inference methods in IDP^3. The general approach consisted of translating inference tasks in the context of a dynamic domain to existing inference methods. Afterwards, we compared IDP^3 to other formalisms and systems and conclude that IDP^3 is one of the few systems supporting this much inferences on dynamic specifications.

Integrating CTL and LTL verification algorithms is a topic for future work.

References

Alviano, M., Calimeri, F., Charwat, G., Dao-Tran, M., Dodaro, C., Ianni, G., Krennwallner, T., Kronegger, M., Oetsch, J., Pfandler, A., Pührer, J., Redl, C., Ricca, F., Schneider, P., Schwengerer, M., Spendier, L. K., Wallner, J. P., and Xiao, G. 2013. The fourth answer set programming competition: Preliminary report. In *LPNMR*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 42–53.

Amadini, R., Gabbrielli, M., and Mauro, J. 2013. Features for building CSP portfolio solvers. arXiv:1308.0227 [cs.AI].

Amanda, C., Andrew, C., Olaya, A. G., Jimenez, S., Lopez, C. L., Sanner, S., and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine 33,* 1 (June), 1–8.

Apt, K. R. 2003. *Principles of Constraint Programming.* Cambridge University Press.

Bogaerts, B. 2014. PacManID: an implementation of Pac-Man using the simulation inference. `http://dtai.cs.kuleuven.be/krr/files/software/various/pacman.tar.gz`.

Bonner, A. J., Kifer, M., and Consens, M. 1993. Database programming in transaction logic. In *In Proc. 4th Int. Workshop on Database Programming Languages*. 309–337.

Calimeri, F., Ianni, G., Ricca, F., Alviano, M., Bria, A., Catalano, G., Cozza, S., Faber, W., Febbraro, O., Leone, N., Manna, M., Martello, A., Panetta, C., Perri, S., Reale, K., Santoro, M. C., Sirianni, M., Terracina, G., and Veltri, P. 2011. The third answer set programming system competition: Preliminary report of the system competition track. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR).* Springer, 388–403.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. 2002. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002).* LNCS, vol. 2404. Springer, Copenhagen, Denmark.

De Cat, B., Bogaerts, B., Bruynooghe, M., and Denecker, M. 2014. Predicate logic as a modelling language: The IDP system. *CoRR abs/1401.6312.*

De Cat, B., Denecker, M., and Stuckey, P. 2012. Lazy model expansion by incremental grounding. In *Proceedings of the 28th International Conference on Logic Programming - Technical Communications (ICLP'12)*, A. Dovier and V. Santos Costa, Eds. LIPIcs, vol. 17. Schloss Daghstuhl - Leibniz-Zentrum fuer Informatik, 201–211.

De Pooter, S., Wittocx, J., and Denecker, M. 2011. A prototype of a knowledge-based programming environment. *CoRR abs/1108.5667.*

Denecker, M. 2012. The FO(·) knowledge base system project: An integration project (invited talk). In *ASPOCP*.

Denecker, M. and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL) 9,* 2 (Apr.), 14:1–14:52.

Denecker, M. and Vennekens, J. 2008. Building a knowledge base system for an integration of logic programming and classical logic. See García de la Banda and Pontelli (2008), 71–76.

Denecker, M. and Vennekens, J. 2014. The well-founded semantics is the principle of inductive definition, revisited. In *KR*. AAAI Press. Accepted.

Fitting, M. 1996. *First-order logic and automated theorem proving (2nd ed.).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

García de la Banda, M. and Pontelli, E., Eds. 2008. *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings.* LNCS, vol. 5366. Springer.

Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., and Schaub, T. 2012. Stream reasoning with answer set programming: Preliminary report. In *KR*, G. Brewka, T. Eiter, and S. A. McIlraith, Eds. AAAI Press, 613–617.

Gelfond, M. and Lifschitz, V. 1998. Action languages. *Electron. Trans. Artif. Intell. 2*, 193–210.

Ghallab, M., Isi, C. K., Penberthy, S., Smith, D. E., Sun, Y., and Weld, D. 1998. PDDL - The Planning Domain Definition Language. Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Green, T. J., Aref, M., and Karvounarakis, G. 2012. Logicblox, platform and language: A tutorial. In *Datalog*, P. Barceló and R. Pichler, Eds. LNCS, vol. 7494. Springer, 1–8.

Haufe, S., Schiffel, S., and Thielscher, M. 2012. Automated verification of state sequence invariants in general game playing. *Artif. Intell. 187*, 1–30.

IDP 2013. The IDP system. `http://dtai.cs.kuleuven.be/krr/software`.

IDPDraw 2012. IDPDraw: finite structure visualization. `http://dtai.cs.kuleuven.be/krr/software/visualisation`.

Ierusalimschy, R., de Figueiredo, L. H., and Celes, W. 1996. Lua – an extensible extension language. *Software: Practice and Experience 26,* 6, 635–652.

Kaufmann, M., Moore, J. S., and Manolios, P. 2000. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Norwell, MA, USA.

Kleene, S. C. 1938. On notation for ordinal numbers. *The Journal of Symbolic Logic 3,* 4, pp. 150–155.

Kowalski, R. A. and Sadri, F. 2013. Towards a logic-based unifying framework for computing. *CoRR abs/1301.6905.*

Kowalski, R. A. and Sergot, M. J. 1986. A logic-based calculus of events. *New Generation Computing 4,* 1, 67–95.

Leuschel, M. and Butler, M. J. 2008. ProB: An automated analysis toolset for the B method. *STTT 10,* 2, 185–203.

Lin, F. and Reiter, R. 1997. How to progress a database. *Artif. Intell. 92,* 1-2, 131–167.

Mariën, M., Gilis, D., and Denecker, M. 2004. On the relation between ID-Logic and Answer Set Programming. In *JELIA*, J. J. Alferes and J. A. Leite, Eds. LNCS, vol. 3229. Springer, 108–120.

Markov, A. A. 1906. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete 2-ya seriya, 15*, 135–156.

The Coq development team. 2004. *The Coq proof assistant reference manual.* LogiCal Project. Version 8.0.

McCarthy, J. and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, B. Meltzer and D. Michie, Eds. Edinburgh University Press, 463–502.

Nemhauser, G. L. and Wolsey, L. A. 1988. *Integer and Combinatorial Optimization.* John Wiley and Sons, New York.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press.

Riazanov, A. and Voronkov, A. 2002. The design and implementation of vampire. *AI Communications 15,* 2-3, 91–110.

Shlyakhter, I., Seater, R., Jackson, D., Sridharan, M., and Taghdiri, M. 2003. Debugging overconstrained declarative models using unsatisfiable cores. In *ASE*. IEEE Computer Society, 94–105.

Sutcliffe, G. 2009. The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. *Journal of Automated Reasoning 43,* 4, 337–362.

Sutcliffe, G. 2013. The 6th IJCAR Automated Theorem Proving System Competition - CASC-J6. *AI Communications 26,* 2, 211–223.

SUTCLIFFE, G., SCHULZ, S., CLAESSEN, K., AND BAUMGARTNER, P. 2012. The TPTP typed first-order form with arithmetic. In *LPAR*, N. Bjørner and A. Voronkov, Eds. Lecture Notes in Computer Science, vol. 7180. Springer, 406–419.

THIELSCHER, M. 2011. A unifying action calculus. *Artif. Intell. 175,* 1 (Jan.), 120–141.

VAN GINKEL, N. 2013. Pddl2IDP: a PDDL parser for IDP. `http://dtai.cs.kuleuven.be/krr/files/software/various/pddl2idp.zip`.

VARDI, M. Y. 1986. Querying logical databases. *Journal of Computer and System Sciences 33,* 2, 142 – 160.

VENNEKENS, J., GILIS, D., AND DENECKER, M. 2006. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic 7,* 4, 765–797.

WEIDENBACH, C., DIMOVA, D., FIETZKE, A., KUMAR, R., SUDA, M., AND WISCHNEWSKI, P. 2009. Spass version 3.5. In *CADE*, R. A. Schmidt, Ed. LNCS, vol. 5663. Springer, 140–145.

WITTOCX, J., MARIËN, M., AND DENECKER, M. 2008. The IDP system: a model expansion system for an extension of classical logic. In *LaSh*, M. Denecker, Ed. 153–165.

Appendix A Weak Progression

In this section, we show how a weaker variant of progression can be defined using three-valued logic. We will restrict our attention to function-free vocabularies (i.e.,vocabularies containing only constant and predicate symbols) here to simplify the presentation. However, all definitions can be extended to the general case.

A.1 Three-valued logic

We briefly summarise some concepts from three-valued logic. A truth-value is one of the following: $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ (true, false and unknown). We define $\mathbf{f}^{-1} = \mathbf{t}, \mathbf{t}^{-1} = \mathbf{f}$ and $\mathbf{u}^{-1} = \mathbf{u}$. We define two orders on truth values: the precision order $\leq_p$ is given by $\mathbf{u} \leq_p \mathbf{t}$ and $\mathbf{u} \leq_p \mathbf{f}$. And the truth order $\leq$ is given by $\mathbf{f} \leq \mathbf{u} \leq \mathbf{t}$.

Definition Appendix A.1
A *partial set* $\mathcal{P}$ over D is a function from D to $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$.

The precision order is extended to partial sets over D: $\mathcal{P} \leq_p \mathcal{P}'$ if for all $d \in D$: $\mathcal{P}(d) \leq_p \mathcal{P}'(d)$.

A partial Σ-structure J consists of 1) a *domain*, D^J: a set of elements, and 2) a mapping associating a value to each symbol in Σ. For predicate symbols P of arity n, this is a partial set P^J over $(D^J)^n$. For constants, this is a value in D^J.

We assume that a (partial) structure also interprets variable symbols and denote $J[x : d]$ for the structure equal to J except interpreting x by d.

If J and J' are two partial structures with the same interpretation for constants, J is less precise than J' ($J \leq_p J'$) if for all symbols σ, $\sigma^J \leq_p \sigma^{J'}$. A partial structure J is two-valued if interpretations of its symbols map nothing to $\mathbf{u}$. A two-valued partial structure is exactly a structure.

Definition Appendix A.2
Given a partial structure J, the Kleene valuation (Kl_J) is defined inductively based on the Kleene truth tables (Kleene 1938):

- $Kl_J(P(\bar{t})) = P^J(\bar{t}^J)$,
- $Kl_J(\neg\varphi) = (Kl_J(\varphi))^{-1}$
- $Kl_J(\varphi \wedge \psi) = \min_{\leq} (Kl_J(\varphi), Kl_J(\psi))$
- $Kl_J(\varphi \vee \psi) = \max_{\leq} (Kl_J(\varphi), Kl_J(\psi))$
- $Kl_J(\forall x : \varphi) = \min_{\leq} \left\{Kl_{J[x:d]}(\varphi) \mid d \in D^J\right\}$
- $Kl_J(\exists x : \varphi) = \max_{\leq} \left\{Kl_{J[x:d]}(\varphi) \mid d \in D^J\right\}$

The Kleene valuation is extended to definitions and theories. For definitions, intuitively, the value of Δ is true if all its defined atoms are two-valued and have the correct (defined) interpretation, its value is false if some defined atom is interpreted incorrectly, and is unknown otherwise. The exact definition can be found in (Denecker and Ternovska 2008). In this text, we will only use the following property.

Proposition Appendix A.3
If all defined atoms in a non-empty definition Δ are interpreted as $\mathbf{u}$ in J, then $Kl_J(\Delta) = \mathbf{u}$.

We use $Kl_J(\mathcal{T})$ to denote the Kleene value of a theory $\mathcal{T}$ over a structure J. $Kl_J(\mathcal{T}) = \mathbf{t}$ if all of $\mathcal{T}$'s definitions and sentences have value $\mathbf{t}$ in structure J. $Kl_J(\mathcal{T}) = \mathbf{f}$ if one of $\mathcal{T}$'s definitions or sentences has value $\mathbf{f}$ in structure J. $Kl_J(\mathcal{T}) = \mathbf{u}$ otherwise.

We summarise some well-known properties about the Kleene-valuation.

Proposition Appendix A.4
If J is a two-valued partial structure (i.e., a structure), then $Kl_J(\mathcal{T})$ is $\mathbf{t}$ if and only if $J \models \mathcal{T}$ and $Kl_J(\mathcal{T})$ is $\mathbf{f}$ otherwise.

Proposition Appendix A.5
If J and J' are partial structures with $J \leq_p J'$, then for every theory $\mathcal{T}$, $Kl_J(\mathcal{T}) \leq_p Kl_{J'}(\mathcal{T})$.

A.2 Weakly $\mathcal{T}$-Compatible Chains and Weak Progression

For this paper, we are only interested in a special kind of partial structures: partial structures that have complete information on an initial segment of time points and that have no information about other time points. Using the identification of a structure with an ∞-chain, a k-chain corresponds to such a partial structure. If $(J_j)_{j=0}^k$ is a k-chain, we associate to J the partial structure J equal to the J_j on static symbols and such that for dynamic symbols σ

$$\sigma(d_1 \ldots d_{n-1}, j)^J = \begin{cases} \mathbf{t} \text{ if } j \leq k \text{ and } (d_1 \ldots d_{n-1}) \in \sigma_{curr}^{J_j} \\ \mathbf{f} \text{ if } j \leq k \text{ and } (d_1 \ldots d_{n-1}) \notin \sigma_{curr}^{J_j} \\ \mathbf{u} \text{ otherwise} \end{cases}$$

We identify the k-chain and the corresponding partial structure.

Definition Appendix A.6 (*Weakly $\mathcal{T}$-compatible, weak $\mathcal{T}$-successor*)
A k-chain J is *weakly $\mathcal{T}$-compatible* with a Σ-theory $\mathcal{T}$ if $Kl_J(\mathcal{T}) \neq \mathbf{f}$.

A Σ_{ss}-structure S' is a *weak $\mathcal{T}$-successor* of a k-chain J if J::S is weakly $\mathcal{T}$-compatible.

Proposition Appendix A.7
Every $\mathcal{T}$-compatible k-chain J is also weakly $\mathcal{T}$-compatible.

Proof
If J is $\mathcal{T}$-compatible, then there is a model J' of $\mathcal{T}$ that is more precise than J. Since $J' \models \mathcal{T}$, $Kl_{J'}(\mathcal{T}) = \mathbf{t}$ by Proposition Appendix A.4. Now, Proposition Appendix A.5 guarantees that $Kl_J(\mathcal{T})$ is less precise than $\mathbf{t}$, hence it must be either $\mathbf{t}$ or $\mathbf{u}$ and we conclude that J is indeed weakly $\mathcal{T}$-compatible. □

The reverse of Proposition Appendix A.7 does not hold as the following (simple) example shows.

Example Appendix A.8
Let $\mathcal{T}$ be the following first-order theory:

$$P(\mathcal{I}).$$
$$\forall t : Q(\mathcal{S}(t)) \Leftrightarrow P(t).$$
$$\forall t : \neg Q(t).$$

It is clear that $\mathcal{T}$ has no models, as the second constraint requires Q to be true at time 1, while the last constraint requires Q to be false at all time points. Hence, there are no $\mathcal{T}$-compatible chains.

However, the 0-chain J such that J_0 interprets P by $\mathbf{t}$ and Q by $\mathbf{f}$ is weakly $\mathcal{T}$-compatible. The Kleene-valuation of $\mathcal{T}$ in J is $\mathbf{u}$.

The above example shows that it is possible that a weakly $\mathcal{T}$-compatible chain cannot be extended. Such a situation is often called a deadlock.

Definition Appendix A.9 (*Deadlock*)
A weakly $\mathcal{T}$-compatible chain J is in a *deadlock* if there are no weakly $\mathcal{T}$-compatible extensions of J.

Definition Appendix A.10 (*Weak Progression inference*)
The *weak progression inference* is an inference that takes as input a theory $\mathcal{T}$ and a weakly $\mathcal{T}$-compatible k-chain J and returns all weak $\mathcal{T}$-successors of J.

Definition Appendix A.11 (*Weak Markov property*)
A theory $\mathcal{T}$ satisfies the *weak Markov property* if for every weakly $\mathcal{T}$-compatible k-chain J, and every weakly $\mathcal{T}$-compatible k'-chain J' ending in the same state, i.e., such that $J_k = J'_{k'}$, the weak $\mathcal{T}$-successors of J are exactly the weak $\mathcal{T}$-successors of J'.

The weak Markov property essentially says the same as the Markov property, namely that the successors of a given chain only depend on the last state, i.e., that the system has no history.

Appendix B Proofs

Proposition 3.6
Let Σ be a linear-time vocabulary and Σ_{ss} the corresponding single state vocabulary. Then the mappings $\pi^{ss}_k(\cdot)$ induce a one-to-one correspondence between Σ-structures J and sequences $(J_k)^{\infty}_{k=0}$ of Σ_{ss}-structures sharing the same interpretation of static symbols.

Proof
It is clear that given a structure J, $(\pi^{ss}_k(J))^{\infty}_{k=0}$ is indeed such a sequence.

Now, for the other direction, suppose J_k is a sequence of Σ_{ss}-structures sharing the same interpretation of static symbols. Let J denote the Σ-structure with the same interpretation of static symbols and such that, for dynamic predicates σ,

$$\sigma^J = \{(d_1, \dots, d_{n-1}, k) \mid (d_1, \dots, d_{n-1}) \in \sigma^{J_k}_{curr}\}.$$

Then J is indeed a structure such that $\pi^{ss}_k(J) = J_k$, as desired. □

Theorem 3.18
Let $\mathcal{T}$ be an LTC-theory and J a Σ-structure. Then J is a model of $\mathcal{T}$ if and only if $\pi^{ss}_0(J) \models \mathcal{T}_0$ and for every $k \in \mathbb{N}$, $\pi^{bs}_k(J) \models \mathcal{T}_t$.

Proof
By the first condition of Definition 3.13, the FO part of the theory only consists of static, initial, single-state, and bistate sentences. Now, a structure J satisfies a static sentence if and only if each of its projections satisfy this sentence. A structure J satisfies an initial sentence, if and only if its initial time-point satisfies the projection of this sentences, etc. Hence, for the FO part, the result easily follows.

Furthermore, Definition 3.13 guarantees that all definitions in $\mathcal{T}$ are stratified over time. Now, it follows immediately from Theorem 4.5 in (Vennekens et al. 2006) that we can split stratified definitions in one definition for each stratification level. Thus, what we obtain is one definition for each point in time, defining the state at $\mathcal{S}(t)$ in terms of the state in t. This definition corresponds exactly to the definition in $\mathcal{T}_t$, as desired. □

Theorem 3.19
Let $\mathcal{T}$ be an LTC-theory and J a k-chain. Then, J is weakly $\mathcal{T}$-compatible if and only if $\pi^{ss}_0(J) \models \mathcal{T}_0$ and for every $j < k$, $\pi^{bs}_j(J) \models \mathcal{T}_t$.

Proof
One direction is clear: if J is weakly $\mathcal{T}$-compatible, then $\pi^{ss}_0(J) \models \mathcal{T}_0$ and for every $j < k$, $\pi^{bs}_j(J) \models \mathcal{T}_t$.

For the other direction, suppose $\pi^{ss}_0(J) \models \mathcal{T}_0$ and for every $j < k$, $\pi^{bs}_j(J) \models \mathcal{T}_t$. We will show that J is weakly $\mathcal{T}$-compatible. In order to show this, we will show that $Kl_J(\mathcal{T}) \neq \mathbf{f}$, or said differently, that for every sentence $\varphi \in \mathcal{T}$, $Kl_J(\varphi) \neq \mathbf{f}$ and that for the definition Δ in $\mathcal{T}$, $Kl_J(\Delta) \neq \mathbf{f}$.

First, let φ be any sentence in $\mathcal{T}$. If φ is an initial, or a static sentence, then $J \models \varphi$ because $\pi^{ss}_0(J) \models \mathcal{T}_0$, thus $Kl_J(\varphi) = \mathbf{t}$ for such sentences. If φ is a universal single-state sentence $\forall t : \varphi'(t)$, we assume that $Kl_J(\varphi) = \mathbf{f}$, and will show that this leads to a contradiction. In this case, using the definition of the Kleene valuation, at least for one i, $Kl_J(\varphi[i/t]) = \mathbf{f}$, or said differently, at least for one i, $J_i \not\models te(\varphi)$. Now, this i should definitely be greater than k, since $\mathcal{T}_t$ contains the constraint $te(\varphi)$. However, since

J_i is completely unknown on dynamic predicates, we see that $J_i \leq_p J_0$. Hence, using Proposition Appendix A.5, we find that also $J_0 \not\models te(\varphi)$, which is in contradiction with the assumption that $\pi_0^{ss}(J) \models \mathcal{T}_0$. For bistate sentences, a similar argument holds. Thus we can conclude that indeed for every sentence φ in $\mathcal{T}$, $Kl_J(\varphi) \neq \mathbf{f}$.

Now, let Δ be the definition of $\mathcal{T}$. We should show that $Kl_J(\Delta) \neq \mathbf{f}$. As Δ is stratified over time, by Theorem 4.5 in (Vennekens et al. 2006), we can split Δ in definitions $(\Delta_i)_{i\in\mathbb{N}}$ for each time point. The definitions Δ_i with $i \leq k$ are satisfied in J because those are the definitions in the theory $\mathcal{T}_t$. For definitions Δ_i with $i > k$, these definitions define only dynamic atoms with dynamic arguments greater than k. Furthermore, these dynamic atoms are completely unknown in J. Proposition Appendix A.3 then yields that $Kl_J(\Delta) = \mathbf{u} \neq \mathbf{f}$.

Thus, we also find that $Kl_J(\mathcal{T}) = \mathbf{u} \neq \mathbf{f}$, i.e., J is indeed weakly $\mathcal{T}$-compatible. □

Corollary 3.20
LTC-theories satisfy the Markov property and the weak Markov property.

Proof of Corollary 3.20
We first prove that LTC theories satisfy the Markov property. Let J and J' be a k-chain and a k'-chain respectively with $J_k = J'_{k'}$. Suppose S is a $\mathcal{T}$-successor of J'. We show that S is also a $\mathcal{T}$-successor of J. Since $J'{::}S$ is $\mathcal{T}$-compatible, there exists a model K' of $\mathcal{T}$ such that $K'_i = J'_i$ for $i \leq k'$ and $K_{k'+1} = S$. Now let K be the structure such that

$$K_j = \begin{cases} J_j & \text{for } j \leq k, \\ K'_{k'+j-k} & \text{otherwise.} \end{cases}$$

We claim that K is a model of $\mathcal{T}$ more precise than $J{::}S$. The fact that it is more precise than $J{::}S$ follows from the fact that $K_{k+1} = K'_{k'+(k+1)-k} = K'_{k'+1}$, which equals S, by construction of K. In order to prove our claim, we show that for every j, (K_j, K_{j+1}), satisfies $\mathcal{T}_t$. For $j \leq k$, this follows from the fact that J is $\mathcal{T}$-compatible; for $j > k$, from the fact that K' is a model of $\mathcal{T}$. Now using Theorem 3.18, we see that K is a model of $\mathcal{T}$, which shows that $J{::}S$ is indeed $\mathcal{T}$-compatible.

We now prove that LTC theories satisfy the weak Markov property. This follows immediately from Theorem 3.19: $J{::}S$ is weakly $\mathcal{T}$-compatible if and only if J is weakly $\mathcal{T}$-compatible and $(J_k, S) \models \mathcal{T}_t$. □

Theorem 4.2
Let $\mathcal{T}$ be an LTC-theory and φ a universal single-state sentence. Then $\mathcal{T} \models \varphi$ if $\mathcal{T}_0 \models te(\varphi)$, and $(\mathcal{T}_t \wedge te(\varphi)) \models te(\varphi[\mathcal{S}(t)/t])$, where t is the unique time-variable in φ.

Proof
This theorem is in fact a reformulation of the principle of proofs by induction. The condition $\mathcal{T}_0 \models te(\varphi)$ expresses that the invariants holds at time 0, i.e., this is the base case. The condition $(\mathcal{T}_t \wedge te(\varphi)) \models te(\varphi[\mathcal{S}(t)/t])$ expresses that whenever the invariants holds at t, it also holds at $\mathcal{S}(t)$. □

Theorem 4.3
Let J be a Σ_s-structure and let φ be a universal single-state sentence with time variable t. Then φ is satisfied in all Σ-structures expanding J if $\mathcal{T}_0 \wedge \neg te(\varphi)$ has no models expanding J, and $\mathcal{T}_t \wedge te(\varphi) \wedge \neg te(\varphi[\mathcal{S}(t)/t])$ has no models expanding J.

Proof
This theorem is also a reformulation of the principle of proofs by induction, analogue to Theorem 4.2. □

Theorem 4.4
Let $\mathcal{T}$ be an LTC-theory and φ a universal bistate sentence. Then $\mathcal{T} \models \varphi$ if and only if $\mathcal{T}_t \models te(\varphi)$.

Proof
One direction, is clear: if $\mathcal{T} \models \varphi$, it follows immediately that $\mathcal{T}_t \models te(\varphi)$.

For the other direction, suppose $\mathcal{T}_t \models te(\varphi)$. We should show that $\mathcal{T} \models \varphi$. Therefore, let J be a model of $\mathcal{T}$. By Theorem 3.18, for every k, $J_k \models \mathcal{T}_t$. Thus, using our assumption, for every k, also $J_k \models te(\varphi)$. But φ is itself an LTC-theory, and $\varphi_i = \mathbf{t}$ and $\varphi_t = te(\varphi)$. Thus, using Theorem 3.18 again, we find that $J \models \varphi$, as desired. □