PITL2MONA: Implementing a Decision Procedure for Propositional Interval Temporal Logic

Rodolfo Gómez[1] — **Howard Bowman**

Computing Laboratory
University of Kent at Canterbury (United Kingdom)
{rsg2,hb5}@kent.ac.uk

ABSTRACT. *Interval Temporal Logic (ITL) is a finite-time linear temporal logic with applications in hardware verification, temporal logic programming and specification of multimedia documents. Due to the logic's non-elementary complexity, efficient ITL-based verification tools have been difficult to develop, even for propositional subsets. MONA is an efficient implementation of an automata-based decision procedure for the logic WS1S. Despite the non-elementary complexity of WS1S, MONA has been successfully applied in problems such as hardware synthesis, protocol verification and theorem proving. Here we consider a rich propositional subset of ITL, PITL, whose expressive power is equivalent to that of WS1S, and in turn to that of regular languages. PITL not only includes operators such as chop (**;**), star ($*$) and projection (**proj**), but also past operators such as previous ($\ominus$), chop in the past ($\widetilde{;}$) and since ($\mathcal{S}$). We provide an interpretation of PITL formulas in WS1S, which led us to a direct translation from PITL formulas to MONA specifications. We present the tool PITL2MONA as an implementation of such translation. With PITL2MONA acting as a front-end, MONA is used as a decision procedure for PITL. To our knowledge, this is one of the few implementations of a decision procedure for PITL, the first one based on automata, and the only one which handles both projection and past operators. We have tested our implementation on a number of examples; we show in this paper the application of PITL and its MONA-based decision procedure in solutions to the dining-philosophers and a multimedia synchronisation problem, together with some experimental results on these and some other examples.*

KEYWORDS: *Interval Temporal Logic, decision procedure, MONA, WS1S.*

1. The author is supported by the ORS Award Scheme, UK Universities.

1. Introduction

Interval Temporal Logic (ITL) [MOS 83] is a linear temporal logic over finite time. ITL (and some of its propositional subsets) have been applied in different problems, from specification and verification of hardware devices [HAL 83], [MOS 83], [MOS 85] and temporal logic programming [MOS 86], [DUA 96] to the specification of multimedia documents [BOW 03a] and human computer interaction [BOW 98a], [BOW 99]. Also, interest in ITL comes from its natural notation and expressiveness. Operators such as *chop*, *projection* and *star* support sequential composition, multiple time granularities and repetitive behaviour in system specifications. Also, high-level, imperative-like operators such as loops, conditionals and assignments can easily be defined, and so ITL naturally lends itself to execution.

ITL's features make this logic an attractive alternative to the problems faced by conventional point-based temporal logics. It is accepted that the specification of properties in such point-based temporal logics could be difficult for non-temporal logic experts. Thus, successful verification of a misformulated property may give unjustified confidence in a system design. In general, for this and other reasons, it has been recognised that specification languages need the full power of regular expressions [PNU 85], [VAR 01]. And it is known that *chop* and *star* bring this expressive power to ITL. Furthermore, there is evidence of increasing industrial interest in ITL. For example, specification languages used in model checking tools such as Verisity's *Temporal e* language [HOL 01] and IBM's *Sugar* [BEE 01], have already introduced ITL concepts. Here we consider a rich propositional subset of ITL over finite intervals, PITL, which has the same expressive power as *Quantified Propositional Temporal Logic* (QPTL) for finite time, regular expressions and *Weak Monadic Second-Order Theory of One Successor* (WS1S) [MOS 00b]. PITL not only includes operators such as *chop* (**;**), *star* (*) and *projection* (**proj**), but also past operators such as *previous* ($\ominus$), *chop in the past* ($\widetilde{;}$) and *since* ($\mathcal{S}$). Past operators do not add expressive power w.r.t. a set of future operators, but they make the logic easier to use.

In recent years there has been substantial research on decision procedures for ITL (and sublogics) [BOW 98b, BOW 03b, MOS 00b, MOS 00a], but practical applications have been limited by the worst-case, non-elementary complexity of ITL [MOS 83]. However, efficient tools for non-elementary logics are now available which have shown that a number of interesting problems and applications do not necessarily fall in this worst-case behavior, and therefore can be handled satisfactorily.

One such tool is MONA [KLA 01, KLA 02], which implements an efficient decision procedure for WS1S [BÜC 60, ELG 61, THO 90]. While it was shown that the decision problem for WS1S is non-elementary [MEY 75], we are interested in this logic because it is as expressive as regular languages (and so PITL can be reduced to it) and also because an efficient implementation for its decision procedure is available (MONA). This evidence speaks in favour of the applicability of non-elementary logics.

This paper explores the connection between PITL and WS1S/MONA. We provide an interpretation of PITL formulas in WS1S, which led us to a direct translation from PITL formulas to MONA specifications. We present the tool PITL2MONA as an implementation of such translation. With PITL2MONA acting as a front-end, MONA is used as a decision procedure for PITL. To our knowledge, this is one of the few implementations of a decision procedure for PITL, the first one based on automata, and the only one which handles both *projection* and past operators.

Related Work.

Proof systems and decision procedures for ITL have been the focus of most of the research done so far. However, the tableau-based tool LITE [KON 95] is the only other implementation of a decision procedure for a propositional subset of ITL. It is also worth saying that while research has also been performed in other related interval logics [PAE 88, HAL 91, VEN 91, DUT 95, CHA 91], to our knowledge *Quantified Discrete-time Duration Calculus* (QDDC) is the only one of them with an implemented decision procedure (DCVALID [PAN 00b]).

Kono's LITE is an implementation of a tableau-based decision procedure for a propositional subset of ITL. LITE supports *chop*, *star*, *projection*, quantification over atomic propositions and a number of high-level, imperative-like iteration operators such as *while*. It does not support, however, past operators. We believe that efficiency is the main advantage of our implementation w.r.t. LITE. Compared with an automata-based decision procedure such as MONA, tableau-based implementations are likely to suffer from the overhead associated with expansion rules and normal form generation. Moreover, MONA includes a number of optimisations which allow even worst-case, non-elementary formulas to be decided with reasonable time and memory requirements [KLA 02]. Also, automata are more compositional than tableau [MOS 00a]. For example, and unlike automata, given two PITL formulas φ and ψ decided by tableau $\mathcal{T}(\varphi)$ and $\mathcal{T}(\psi)$, it is difficult to reuse them to obtain $\mathcal{T}(\varphi \,;\, \psi)$.

Pandya's tool DCVALID [PAN 00b, PAN 00a] implements a decision procedure for QDDC, a quantified discrete-time subset of *Duration Calculus* [CHA 91]. DCVALID is also implemented very much as a front-end which translates QDDC formulas to MONA specifications. QDDC is closely related to PITL; DCVALID supports *chop* and *star*-like operators. It also implements quantification over atomic propositions, and a *duration* operator which counts the number of times a given proposition is true in the interval (this operator is a distinctive feature of QDDC). However, DCVALID does not support any form of *projection*, and only a limited form of the past operator *previous* (this operator can only be applied over atomic propositions).

Neither *projection* nor quantification add expressive power to a propositional subset of ITL containing both *chop* and *star* which, as mentioned before, is already as expressive as regular languages. However, and depending on the application domain, these operators can make the logic easier to use. As WS1S also has the expressive power of regular languages, it turns out that both *projection* and quantification can be interpreted in WS1S. Nevertheless we have only included *projection* in our subset,

for a number of reasons. First, since our main objective is to show that a translation from a rich subset of PITL to WS1S is feasible, the implementation of a full set of PITL operators would fall out of the paper's scope. Secondly, while quantification over PITL atomic propositions can be directly interpreted as WS1S second-order quantification (Section 2 will show that PITL atomic propositions can be interpreted as WS1S second-order variables), the interpretation of *projection* in WS1S is not so straightforward and thus it better shows the power of WS1S to encode PITL. Also, and to the best of our knowledge, a definition of *projection* using quantification has not been presented in the literature (and it seems rather difficult to obtain!). Finally, *projection* has already found important applications in different contexts. For example, *star* and other iteration operators of the imperative *while* and *for* loop variety can be defined in terms of *projection* [BOW 03b]. Also, it can be used to realise temporal abstraction in the real-time setting [MEL 93], and hence, for example, to describe the slowing down of multimedia presentations [BOW 03a] (Section 7 will illustrate such application). And in other disciplines, *projection* has been found to be useful for describing goals for cognitive behaviour in the domain of multi-modal human computer interaction [BOW 98a, BOW 99].

Paper outline.

Sections 2 and 3 give the necessary overview on the logics PITL, WS1S and the MONA tool. Section 4 shows an interpretation of PITL formulas (without past operators) in WS1S. Section 5 shows the translation algorithm (sketched) from PITL formulas to MONA specifications. In section 6 we include past operators in the language, and show how semantics and translation can be easily adapted. In section 7 both, a version of the well-known dining philosophers problem and a multimedia synchronisation problem serve as examples of specification and verification of systems using PITL. Some implementations details of the tool PITL2MONA, together with experimental results are shown in section 8. Here the performance of PITL2MONA is compared with that of LITE (Kono's tableau-based implementation). Conclusions are discussed in section 9. Appendix A defines satisfiability for the whole set of PITL operators (future and past operators); theorems and their proofs can be found in appendix B.

2. Propositional Interval Temporal Logic

PITL is defined over finite state sequences. Each sequence is called an interval and $\mathcal{I}$ denotes the set of all possible intervals; $\sigma \in \mathcal{I}$ has the form:

$$\langle \sigma_0, \sigma_1, \ldots, \sigma_{|\sigma|} \rangle$$

where $|\sigma|$ denotes the length of an interval and σ_i denotes the i-th state in an interval. By convention, the length of an interval is the number of states minus one and all

intervals must have at least one state. We use $[\sigma]^i$ ($(\sigma)^i$) to denote the i-th prefix (suffix) of an interval. Formally,

$$[\sigma]^i = \langle \sigma_0, \ldots, \sigma_i \rangle \qquad (\sigma)^i = \langle \sigma_i, \ldots, \sigma_{|\sigma|} \rangle$$

Intervals are interpreted w.r.t. a finite set of atomic propositions $\mathcal{A} = \{v_1, \ldots, v_m\}$. Each state $\sigma_i \in \mathbb{P}(\mathcal{A})$ is a set containing all atomic propositions that are considered true at that state. In this section we will just present the syntax and semantics of *PITL*^{1}, a fragment of PITL which does not handle past operators (syntax and semantics of past operators are given in Section 6). A *PITL*^{1} formula F is constructed as follows ($v \in \mathcal{A}$):

$$F ::= v \mid \textbf{False} \mid \neg F \mid F \vee F \mid \textbf{empty} \mid \bigcirc F \mid F \textbf{ ; } F \mid F^* \mid F \textbf{ until } F \mid F \textbf{ proj } F$$

We introduce a satisfaction relation, $\models$, to interpret *PITL*^{1} formulas over intervals: $\sigma \models F$ denotes that σ satisfies (or is a model for) the formula F. The reader familiar with ITL will notice that our definitions for *star* and *projection* are slightly different from those originally provided by Moszkowski (see e.g. [MOS 86], [MOS 00a]). It turns out that both semantics are equivalent: they describe exactly the same models. However, the translation to WS1S is very much simplified if we rule out empty iterations. We prove the equivalence only for *star* (see Theorem 10 in appendix B), but the proof for *projection* can be easily derived in the same way. We define the semantics of *PITL*^{1} by induction over the structure of formulas, as follows ($v \in \mathcal{A}$, P, Q are *PITL*^{1} formulas):

$\sigma \models v$ iff $v \in \sigma_0$

$\sigma \not\models \textbf{False}$

$\sigma \models \neg P$ iff $\sigma \not\models P$

$\sigma \models P \vee Q$ iff $\sigma \models P \ \vee \ \sigma \models Q$

$\sigma \models \textbf{empty}$ iff $|\sigma| = 0$

$\sigma \models \bigcirc P$ iff $|\sigma| > 0 \ \wedge \ (\sigma)^1 \models P$

$\sigma \models P \textbf{ ; } Q$ iff $\exists k \in \mathbb{N}.\, k \leq |\sigma| \ \wedge \ [\sigma]^k \models P \ \wedge \ (\sigma)^k \models Q$

$\sigma \models P^*$ iff $|\sigma| = 0 \ \vee \ \exists k_0, k_1, \ldots, k_m \in \mathbb{N}.$
$k_0 = 0 < k_1 < \ldots < k_m = |\sigma| \wedge$
$\forall i \in \mathbb{N}.\, 0 \leq i < m \Rightarrow ([\sigma]^{k_{i+1}})^{k_i} \models P$

$$\sigma \models P \textbf{ until } Q \quad \text{iff} \quad \exists k \in \mathbb{N}.\, k \leq |\sigma| \wedge (\sigma)^k \models Q \wedge \forall j \in \mathbb{N}.\, j < k \Rightarrow (\sigma)^j \models P$$

$$\sigma \models P \textbf{ proj } Q \quad \text{iff} \quad |\sigma| = 0 \vee \exists k_0, k_1, \ldots, k_m \in \mathbb{N}.\ k_0 = 0 < k_1 < \ldots < k_m = |\sigma| \wedge \forall i \in \mathbb{N}.\, 0 \leq i < m \Rightarrow ([\sigma]^{k_{i+1}})^{k_i} \models P \wedge \langle \sigma_{k_0}, \sigma_{k_1}, \ldots, \sigma_{k_m} \rangle \models Q$$

Figs. 1 to 5 will help the reader understand the behaviour of *PITL*^{1} operators. $\bigcirc P$ holds over σ if P holds over the suffix $(\sigma)^1$ (Fig. 1). P **;** Q holds over σ if a $0 \leq k \leq |\sigma|$ can be found s.t. P holds over the prefix $[\sigma]^k$ and Q holds over the suffix $(\sigma)^k$ (Fig. 2). P^* holds over σ if either σ is an empty interval or a sequence $0 = k_0 < k_1 < \ldots < k_m \leq |\sigma|$ can be found s.t. it partitions σ into a sequence of subintervals $([\sigma]^{k_{i+1}})^{k_i},\ 0 \leq i < m$ and P holds over each subinterval (Fig. 3). P **until** Q holds over σ if $0 \leq k \leq |\sigma|$ can be found s.t. Q holds over the suffix $(\sigma)^k$ and P holds in every suffix $(\sigma)^j,\ j < k$ (Fig. 4). P **proj** Q holds over σ if either σ is an empty interval or a sequence $0 = k_0 < k_1 < \ldots < k_m \leq |\sigma|$ can be found s.t. it partitions σ into a sequence of subintervals $([\sigma]^{k_{i+1}})^{k_i},\ 0 \leq i < m$, P holds over each subinterval and Q holds on the interval formed by "glueing" together these subintervals' end points, i.e. over $\sigma' = \langle \sigma_{k_0}, \sigma_{k_1}, \ldots, \sigma_{k_m} \rangle$ (Fig. 5).

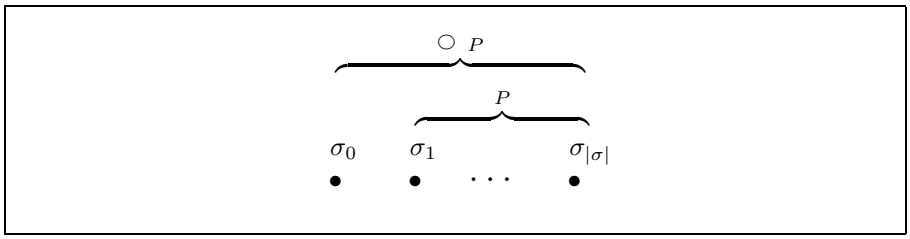

Figure 1. *The next operator*

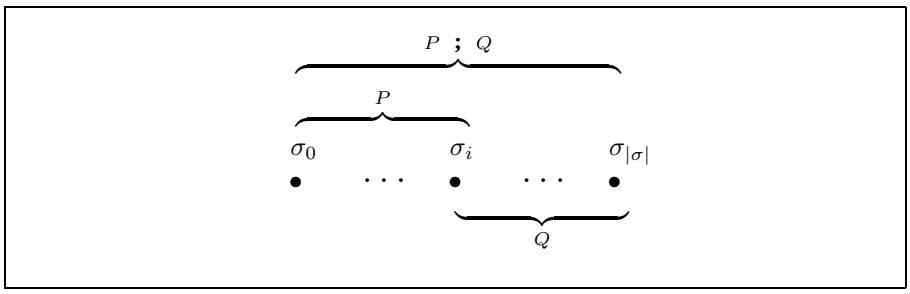

Figure 2. *The chop operator*

Other commonly-used operators can be derived from the basic set, as shown below (P, Q are *PITL*^{1} formulas, $n \in \mathbb{N}$):

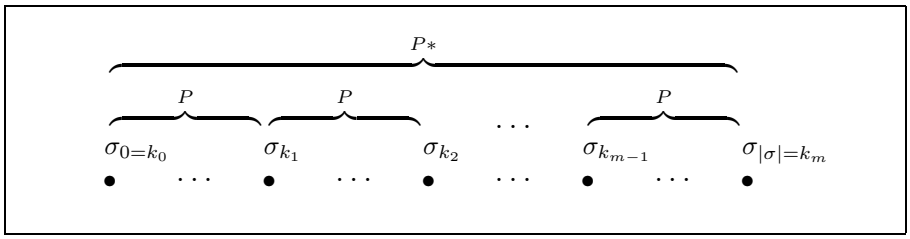

Figure 3. *The star operator*

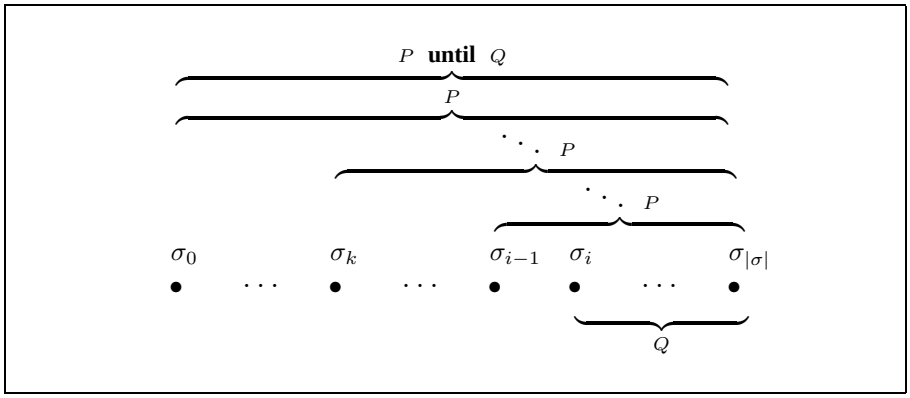

Figure 4. *The until operator*

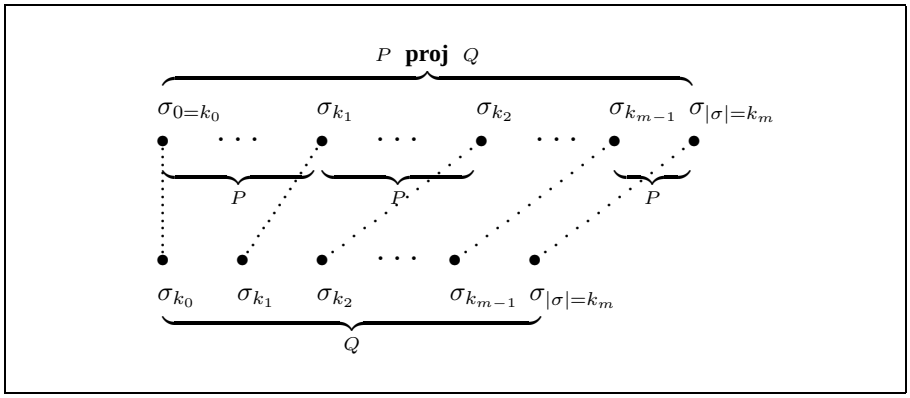

Figure 5. *The projection operator*

$P \wedge Q$	$\equiv \neg(\neg P \vee \neg Q)$	(conjunction)
$P \Rightarrow Q$	$\equiv \neg P \vee Q$	(implication)
$P \Leftrightarrow Q$	$\equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$	(equivalence)
$\Diamond P$	$\equiv$ **True ;** P	(eventually)
$\Box P$	$\equiv \neg\Diamond \neg P$	(always)
halt(P)	$\equiv \Box(P \Leftrightarrow$ **empty**$)$	(halt)
len(n)	$\equiv \underbrace{\bigcirc\bigcirc\cdots\bigcirc}_{n}$**empty**	(length)

3. Weak Monadic Second-order Theory of One Successor (WS1S) and MONA

This section will give the necessary background on WS1S and MONA, mainly taken from [KLA 01]. WS1S [BÜC 60, ELG 61, THO 90] is a decidable logic with an interpretation tied to arithmetic. WS1S formulas are constructed over first-order and second-order variables. Let ϕ denote a WS1S formula, p, q two first-order variables and X a second-order variable. The syntax of WS1S formulas is given by the following set of operators:

$$\phi ::= p = q + 1 \mid p \in X \mid \neg\phi \mid \phi \vee \phi \mid \exists p.\ \phi \mid \exists X.\ \phi$$

WS1S is interpreted over $\mathbb{N}$; first-order variables range over natural numbers, second-order variables range over finite sets of natural numbers and operators $=$,$+$, $\in$, $\neg$, $\vee$ and $\exists$ have the classic interpretation. Other operators can be derived from these, which are shown below[1] (ϕ, ψ denote WS1S formulas, $0, n \in \mathbb{N}$, $p, q, r, z_0, \ldots, z_n$ denote first-order variables, and X, Y, Z denote second-order variables):

$$\begin{array}{lcl}
\phi \wedge \psi & \equiv & \neg(\neg\phi \vee \neg\psi) \\
\phi \Rightarrow \psi & \equiv & \neg\phi \vee \psi \\
\phi \Leftrightarrow \psi & \equiv & (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi) \\
\forall p.\ \phi & \equiv & \neg\exists p.\ \neg\phi \\
\forall X.\ \phi & \equiv & \neg\exists X.\ \neg\phi \\
p = 0 & \equiv & \neg\exists q.\ p = q + 1 \\
p = n & \equiv & \exists z_0, \ldots, z_n.\ z_0 = 0 \wedge z_n = p \wedge \bigwedge_{0 \le i < n} z_{i+1} = z_i + 1 \\
p = q & \equiv & \exists r.\ r = p + 1 \wedge r = q + 1 \\
p + n \in X & \equiv & \exists z_0, \ldots, z_n.\ z_0 = p \wedge z_n \in X \wedge \bigwedge_{0 \le i < n} z_{i+1} = z_i + 1 \\
p \le q & \equiv & \forall X.\ q \in X \wedge (\forall z.\ z + 1 \in X \Rightarrow z \in X) \Rightarrow p \in X \\
p < q & \equiv & p \le q \wedge \neg(p = q) \\
X \subseteq Y & \equiv & \forall p.\ p \in X \Rightarrow p \in Y \\
Z = X \backslash Y & \equiv & \forall p.\ p \in Z \Rightarrow p \in X \wedge \neg(p \in Y) \\
X = Y + 1 & \equiv & \forall p.\ p \in Y \Leftrightarrow p + 1 \in X
\end{array}$$

1. For convenience, the following sections will refer to this extended set simply as WS1S.

MONA [KLA 01] implements a decision procedure for WS1S based on a translation from WS1S formulas to DFA (Deterministic Finite Automaton) [BÜC 60, ELG 61]. The syntax of MONA's input specification language is that of WS1S augmented with syntactic sugar, that is, no expressive power is added. A MONA specification consists of a declaration section and a formula section. Boolean, first-order and second-order variables can be declared. Predicates can also be declared, which instantiate a given formula with actual parameters. Formulas are built using the usual logic connectives, such as $\sim$ (negation), `&` (conjunction), | (disjunction) and `=>` (implication). Expressions on first order variables include relational operators (e.g. `t1>=t2`), addition of constant values (`t+`n) and quantification (`ex1 t:`φ, `all1 t:`φ). Expressions on second-order variables include `min T`, `max T` (minimum and maximum element in a set), `t in T` (membership), `T1 sub T2` (set inclusion), quantification (`ex2 T:`φ, `all2 T:`φ) and other typical set operations like intersection, difference and union. MONA translates a WS1S formula to a minimum DFA that represents the set of satisfying interpretations. Models (counterexamples) of the formula are then expressed by paths from the initial state to an accepting (rejecting) state; MONA returns only the shortest model (counterexample). For example, MONA returns `X={0,1,2}` and `Y={1,2,3}` as a model for the following formula (`X={}` and `Y={}` are respectively returned as a counterexample):

```
var2 X,Y;
X={0,1,3} & all1 k:k in X => k+1 in Y;
```

A conceptual translation from WS1S formulas to DFA [BÜC 60, ELG 61] can be explained in terms of the following simplified WS1S syntax, which do not include first-order variables. It can be shown that this language is as expressive as the original set of operators (ϕ denotes a WS1S formula, X, Y, Z denote second-order variables).

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists X.\, \phi \mid X \subseteq Y \mid X = Y \backslash Z \mid X = Y + 1$$

A string interpretation can be given to finite sets of natural numbers; to the finite set X corresponds any string $s = s_0 s_1 \ldots s_n \in \{0,1\}^*$ (where $s_i,\ 0 \leq i \leq n$ is called a *letter*) such that

$$\forall i.\, 0 \leq i \leq n \wedge i \in X \Leftrightarrow s_i = 1$$

For example, the set $X = \{0,1,3\}$ can be interpreted as the string 1101 (and also as any string $s \in 11010^*$). The semantics are then extended such that a formula with k variables is interpreted over strings $w \in (\{0,1\}^k)^*$. For example, $X = \{0,1,2\}$ and $Y = \{1,2,3\}$ are interpreted as the string:

$$\begin{array}{cccccc} X & \left(\begin{array}{c}1\\0\end{array}\right) & \left(\begin{array}{c}1\\1\end{array}\right) & \left(\begin{array}{c}1\\1\end{array}\right) & \left(\begin{array}{c}0\\1\end{array}\right) \\ Y & & & & & \\ & 0 & 1 & 2 & 3 \end{array}$$

It is assumed that every variable in the formula is assigned a unique index $1, 2, \ldots, k$. Let X_i denote the variable with index $i, 0 \leq i \leq k$. The *projection* of a string w onto X_i is called the X_i-track of w, and $w[M/X_i]$ denotes the shortest string that interprets all variables X_j where $j \neq i$ as w does, but interprets X_i as the set M. A string determines an interpretation $w(X_i)$ of X_i defined as the finite set $\{j|$ the j-th. bit in the X_i-track is $1\}$. Satisfiability is then defined as follows:

$$
\begin{array}{lll}
w \models \neg\phi & \text{iff} & w \not\models \phi \\
w \models \phi \wedge \psi & \text{iff} & w \models \phi \text{ and } w \models \psi \\
w \models \exists X_i.\, \phi & \text{iff} & \text{exists a finite } M \subseteq \mathbb{N} \text{ s.t. } w[M/X_i] \models \phi \\
w \models X_i \subseteq X_j & \text{iff} & w(X_i) \subseteq w(X_j) \\
w \models X_i = X_j \backslash X_k & \text{iff} & w(X_i) = w(X_j) \backslash w(X_k) \\
w \models X_i = X_j + 1 & \text{iff} & w(X_i) = \{j+1 \mid j \in w(X_j)\}
\end{array}
$$

The language $\mathcal{L}(\phi)$ is then defined as the set of satisfying strings $\mathcal{L}(\phi) = \{w \mid w \models \phi\}$. A minimum DFA A_ϕ s.t. $\mathcal{L}(A_\phi) = \mathcal{L}(\phi)$ is constructed inductively on the structure of ϕ: basic, "hand-crafted" automata correspond to atomic formulas, and automata operations are applied to translate composite formulas. Automata defining languages $\mathcal{L}(P \subseteq Q)$, $\mathcal{L}(P = Q \backslash R)$ and $\mathcal{L}(P = Q + 1)$, where P, Q and R are unconstrained second-order variables, are shown in Fig. 6. In these automata, the initial state is also the only accepting state (denoted by a double circle). Also, and just for notational convenience, some transitions are labelled with multiple letters and the symbol x in a letter stands for a component which can be either 0 or 1.

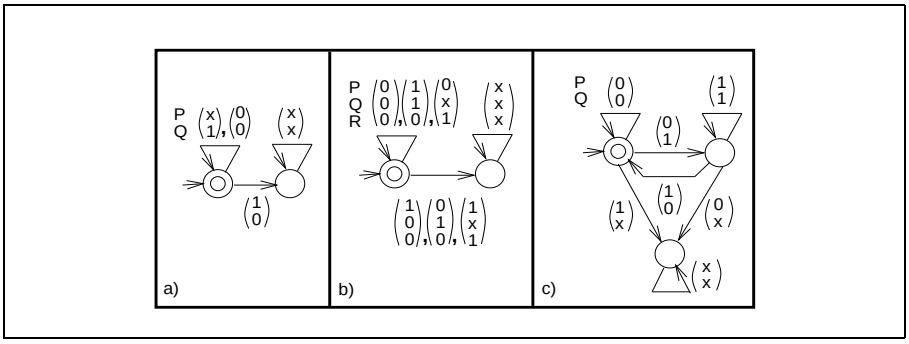

Figure 6. *Automata for a) $\mathcal{L}(P \subseteq Q)$, b) $\mathcal{L}(P = Q \backslash R)$ and c) $\mathcal{L}(P = Q + 1)$*

Negation ($\neg\phi$) corresponds to language complementation ($\overline{\mathcal{L}(\phi)}$) and thus to automata complementation ($\complement A_\phi$). This is a linear-time operation which just flips accepting and rejecting states.

$$\mathcal{L}(\neg\phi) = \overline{\mathcal{L}(\phi)} = \mathcal{L}(A_{\neg\phi}) = \mathcal{L}(\complement A_\phi)$$

Conjunction ($\phi \wedge \psi$) corresponds to language intersection ($\mathcal{L}(A_\phi) \cap \mathcal{L}(A_\psi)$) and thus to automata product ($A_\phi \times A_\psi$); where only the reachable product states are

calculated, i.e. pairs of the form (s_ϕ, s_ψ) where s_ϕ is a state of A_ϕ and s_ψ is a state of A_ψ. This operation may cause a quadratic increase in the automaton size.

$$\mathcal{L}(\phi \wedge \psi) = \mathcal{L}(A_\phi) \cap \mathcal{L}(A_\psi) = \mathcal{L}(A_{\phi \wedge \psi}) = \mathcal{L}(A_\phi \times A_\psi)$$

Second-order existential quantification ($\exists X_i.\ \phi$) corresponds to an application of a right-quotient operation to $\mathcal{L}(\phi)$ followed by a projection operation for X_i over the resulting automaton. These language-operations are defined as follows, where L/L' denotes the right-quotient of a language L with a language L' and $E^i(L)$ denotes the projection of L for X_i.

$$\begin{array}{ll} L/L' & = \{w \mid \exists u \in L'.\ wu \in L\} \\ E^i(L) & = \{w \mid \exists w' \in L.\ w \textit{ is identical to } w' \textit{ except for the } X_i\textit{-track}\} \\ L^i & = \{w \in (\{0,1\}^k)^* \mid \textit{the } X_j\textit{-track } w \textit{ is of the form } 0^* \textit{ for } j \neq i\} \\ \mathcal{L}(\exists X_i.\ \phi) & = E^i(\mathcal{L}(\phi)/L^i) \end{array}$$

Intuitively, $A_{\exists X_i.\ \phi}$ acts as A_ϕ except that it is allowed to "guess" the bits of the X_i-track. This is obtained by a projection operation on A_ϕ which results in a non-deterministic automaton which has to be determinised and minimised. However, before *projection* is applied a right-quotient operation transforms A_ϕ so as to accept just minimal-length strings, i.e. to remove the $(0^k)^*$-suffix (remember, e.g. that models for $X = \{0,1,3\}$ are in 11010^*, with the minimal-length model being 1101). Quantification may cause an exponential increase in the automaton size, due to determinisation.

Meyer [MEY 75] showed that the time and space for translating WS1S formulas to automata, in the worst case, is bounded from below by a stack of exponentials whose height is proportional to the depth of quantifier alternation ($\forall X.\ \exists Y.\ \phi$). In turn, the translation of alternating quantifiers relates to automata determinisation and complementation, which can produce an exponential blow-up ($\forall X.\ \exists Y.\ \phi \equiv \neg\exists X.\ \neg\exists Y.\ \phi$). For example, given $||A_\phi||$, the size of the automaton corresponding to the WS1S formula ϕ, the following holds;

$$\textit{if } ||A_\phi|| = n \textit{ then } ||A_{\neg\exists Y.\ \phi}|| \leq 2^n \textit{ and } ||A_{\neg\exists X.\ \neg\exists Y.\ \phi}|| \leq 2^{2^n}$$

MONA implements the conceptual translation discussed above. A number of syntactic transformations are first applied to a formula in MONA's input specification language to reduce it to a simplified language (equivalent to the WS1S simplified language presented before), where some practical issues such as the interpretation of first-order variables as second-order variables, and the interpretation of boolean variables are considered. For example, the formula $\phi \equiv p = 0$ (where p denotes a first-order variable), could be handled as $\phi' \equiv P = \{0\}$ (where P denotes a second-order variable), but then the formulas $\neg\phi$ and $\neg\phi'$ will lead to different representations (a property expressing that P is a singleton should be conjoined to $\neg\phi'$). MONA encodes

first-order values not as singletons but as non-empty sets (the first-order value corresponds to the smallest element in this set), which is more efficient than the singleton-set approach. Details about these and other issues in the actual translation process can be found in [KLA 01].

Despite the non-elementary complexity of the decision problem, MONA has been applied in many non-trivial situations such as controller synthesis [SAN 98], protocol verification [SMI 00] and theorem proving [OWR 00], [BAS 00] (more links can be found in [KLA 01]). MONA's successful applications can be explained by optimisations performed during the translation process as well as by the fact that the decision procedure is non-elementary in the worst-case, which may not arise so frequently in practice. Optimisations include the use of BDDs to efficiently implement automata (particularly, to provide an efficient implementation of the automaton alphabet and transition function), formula reductions and other techniques to simplify and reuse computations [KLA 02].

4. Interpreting PITL in WS1S

PITL intervals are defined as sets of propositions. From a different point of view (very much influenced by the string-based interpretation of WS1S), we can see propositions as sets of interval states: a state is included in the set if and only if the proposition is true in that state. Formally, for a given interval σ and a proposition $v \in \mathcal{A}$, we define $Set(v, \sigma) = \{i \mid v \in \sigma_i\}$. The following example illustrates this interpretation ($S_i, S \subseteq \mathcal{A}$ interprets state σ_i as a set of propositions).

EXAMPLE 1. — For $\mathcal{A} = \{a, b\}$, $\sigma = \langle \{a\}_0, \{a, b\}_1, \{a, b\}_2, \{b\}_3, \{b\}_4 \rangle$

$$\begin{array}{lccccc} a & 1 & 1 & 1 & 0 & 0 \\ b & 0 & 1 & 1 & 1 & 1 \\ & \sigma_0 & \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 \end{array}$$

$$\begin{array}{lccccccc} Set(a,\sigma) = \{ & 0, & & 1, & 2 & & & \} \\ Set(b,\sigma) = \{ & & & 1, & 2, & 3, & 4 & \} \end{array}$$

□

While this interpretation over sets is sufficient to encode intervals, we will add information representing the current subinterval under consideration. This is very convenient for translating PITL to WS1S formulas, because the semantics of a given PITL formula F over σ is recursively defined in terms of the semantics of subformulas in F over subintervals of σ (where the subintervals are selected by the temporal operators in F). Subintervals of $\sigma = \langle \sigma_0, \ldots, \sigma_{|\sigma|} \rangle$ will be represented by pairs (i, n) of state indices, $0 \leq i \leq n \leq |\sigma|$. This approach has been used before in other temporal logics, see e.g. [MAN 95]. Formally, we can interpret intervals in WS1S as follows. Let $\mathcal{V} = \{V_1, \ldots, V_m\}$ denote a set of WS1S second-order variables, and $\mathbb{FN}$ the set of all finite sets of natural numbers.

DEFINITION 2. — *A $\mathcal{C}$-structure is a tuple $\langle i, n, \mathcal{P}\rangle$ where $i, n \in \mathbb{N},\ i \leq n$ and $\mathcal{P} = \{(V_1, S_1), \ldots, (V_m, S_m)\}$ s.t. $S_j \in \mathbb{FN}$, $1 \leq j \leq m$.*

A $\mathcal{C}$-structure $\langle i, n, \mathcal{P}\rangle$ is intended to model the subinterval $[(\sigma)^i]^n$, $0 \leq i \leq n \leq |\sigma|$ where $\mathcal{P}$ is a set of pairs denoting the atomic propositions in $\mathcal{A} = \{v_1, \ldots, v_m\}$ encoded as second-order variables in $\mathcal{V} = \{V_1, \ldots, V_m\}$; and the states where they are contained encoded as finite sets of natural numbers. Let $\mathcal{C}$ be the set of all $\mathcal{C}$-structures. We define a mapping $\Theta : \mathcal{I} \rightarrow \mathcal{C}$ between intervals and $\mathcal{C}$-structures as follows.

DEFINITION 3. — $\Theta(\sigma) = \langle 0, |\sigma|, \mathcal{P}\rangle$ *where*
$\mathcal{P} = \{(V_i, Set(v_i, \sigma)) \mid v_i \in \mathcal{A},\ V_i \in \mathcal{V},\ 1 \leq i \leq m\}$.

EXAMPLE 4. — For $\mathcal{A} = \{a, b\}$, $\mathcal{V} = \{A, B\}$, and
$\sigma = \langle \{a\}_0, \{a, b\}_1, \{a, b\}_2, \{b\}_3, \{b\}_4\rangle$, the corresponding mapping is given by
$\Theta(\sigma) = \langle 0, 4, \{(A, \{0, 1, 2\}), (B, \{1, 2, 3, 4)\}\rangle$ □

We first define *PITL*^{0}, a fragment of *PITL*^{1} which does not consider *projection*. Formulas in *PITL*^{0} can be interpreted over $\mathcal{C}$-structures as follows. Notice that this interpretation, while somehow free in its notation, is basically using operators which can be expressed in WS1S (see Section 3).

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} v$ iff $(V, S) \in \mathcal{P} \ \wedge\ i \in S$

$\langle i, n, \mathcal{P}\rangle \not\models_{\mathcal{C}}$ **False**

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} \neg P$ iff $\langle i, n, \mathcal{P}\rangle \not\models_{\mathcal{C}} P$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} \bigcirc P$ iff $i < n \ \wedge\ \langle i + 1, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P \vee Q$ iff $\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P \ \vee\ \langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} Q$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}}$ **empty** iff $i = n$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$ **;** Q iff $\exists k.\ i \leq k \leq n \wedge\ \langle i, k, \mathcal{P}\rangle \models_{\mathcal{C}} P \wedge$
$\langle k, n, \mathcal{P}\rangle \models_{\mathcal{C}} Q$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P^*$ iff $\exists k_0, k_1, \ldots, k_m.\ i = k_0 < k_1 < \ldots < k_m = n$
$\wedge\ \forall j.\ 0 \leq j < m \Rightarrow \langle k_j, k_{j+1}, \mathcal{P}\rangle \models_{\mathcal{C}} P$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$ **until** Q iff $\exists k.\ i \leq k \leq n\ \wedge \langle k, n, \mathcal{P}\rangle \models_{\mathcal{C}} Q \wedge$
$\forall j.\ i \leq j < k \Rightarrow \langle j, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$

Satisfaction of *PITL*^{0} formulas over $\mathcal{C}$-structures can be shown to be necessary and sufficient w.r.t. satisfaction over intervals, for both describe exactly the same models (see Theorem 12 in appendix B). Formally,

For any $\sigma \in \mathcal{I}$ and for any PITL^0 formula F, $\Theta(\sigma) \models_{\mathcal{C}} F$ iff $\sigma \models F$

The *projection* operator, however, needs a more expressive structure to be interpreted to WS1S. Basically, $\Theta(\sigma)$ is not enough to interpret a formula like P **proj** Q because Q has to be (recursively) interpreted in a possibly non-continuous subinterval of σ, i.e. the interval which results from "glueing" together only those states of σ which bound the iterations that satisfy P. So a pair of indices (i, n) no longer suffices to encode this new interval; not all state indices between i and n are always included in it. We call this a *sparse* interval. To model this kind of interval, we will extend a $\mathcal{C}$-structure with a set M which represents only those states of σ that must be taken into account to interpret the current subformula.

DEFINITION 5. — *A $\mathcal{C}'$-structure is a tuple $\langle i, n, M, \mathcal{P}\rangle$ where $M \in \mathbb{FN}$, $i, n \in M$, $i \leq n$, and $\mathcal{P} = \{(V_1, S_1), \ldots, (V_m, S_m)\}$ s.t. $S_j \in \mathbb{FN}$, $1 \leq j \leq m$.*

A $\mathcal{C}'$-structure $\langle i, n, M, \mathcal{P}\rangle$ is then intended to model the sparse subinterval $\sigma' = \langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle$, where $i = t_0 < t_1 < \ldots < t_s = n$, $\{t_0, t_1, \ldots, t_s\} = M \cap \{i, \ldots, n\}$. For example, Fig. 7 shows that subinterval $([\sigma]^7)^2$ is a model for formula P **proj** Q, provided both $([\sigma]^7)^2 \models P^*$ and $\langle \sigma_2, \sigma_4, \sigma_6, \sigma_7\rangle \models Q$ hold, where $\{\sigma_2, \sigma_4, \sigma_6, \sigma_7\}$ is the set of states which bound every iteration of P in $([\sigma]^7)^2$. Notice that $\sigma' = \langle \sigma_2, \sigma_4, \sigma_6, \sigma_7\rangle$, the sparse subinterval where Q is required to hold, is built from those states which bound the subintervals where iterations of P hold. Assuming that $\mathcal{P}$ encodes the propositions in σ, formulas P **proj** Q and Q are interpreted over the following $\mathcal{C}'$-structures:

$$\langle 2, 7, \{2, 3, 4, 5, 6, 7\}, \mathcal{P}\rangle \models_{\mathcal{C}'} P \textbf{ proj } Q$$
$$\langle 2, 7, \{2, 4, 6, 7\}, \mathcal{P}\rangle \models_{\mathcal{C}'} Q$$

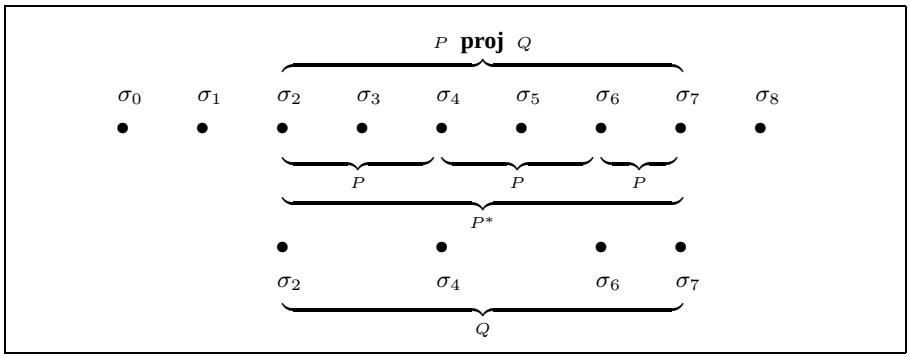

Figure 7. *The projection operator and sparse subintervals*

Let $\mathcal{C}'$ be the set of all $\mathcal{C}'$-structures. We define a mapping $\Theta' : \mathcal{I} \to \mathcal{C}'$ from intervals to $\mathcal{C}'$-structures as follows:

DEFINITION 6. — $\Theta'(\sigma) = \langle 0, |\sigma|, M, \mathcal{P}\rangle$ *where* $\forall i.\ i \in M \Leftrightarrow 0 \leq i \leq |\sigma|$, *and* $\mathcal{P} = \{(V_j, Set(v_j, \sigma)) \mid v_j \in \mathcal{A},\ V_j \in \mathcal{V},\ 1 \leq j \leq m\}$.

We now define the satisfaction relation $\models_{\mathcal{C}'}$ for formulas in *PITL*^{1} as follows:

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} v \quad \text{iff} \quad (V,S)\in\mathcal{P}\wedge i\in S$$

$$\langle i,n,M,\mathcal{P}\rangle \not\models_{\mathcal{C}'} \textbf{False}$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} \neg P \quad \text{iff} \quad \langle i,n,M,\mathcal{P}\rangle \not\models_{\mathcal{C}'} P$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} \bigcirc P \quad \text{iff} \quad \exists j\in M.\ cons(M,i,j)\wedge \langle j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P\vee Q \quad \text{iff} \quad \langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P \ \vee\ \langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} Q$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} \textbf{empty} \quad \text{iff} \quad i=n$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P\ \textbf{;}\ Q \quad \text{iff} \quad \exists k\in M.\ i\le k\le n\wedge \langle i,k,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P\ \wedge\ \langle k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} Q$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P\ \textbf{until}\ Q \quad \text{iff} \quad \exists k\in M.\ i\le k\le n\wedge \langle k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} Q\wedge \forall j\in M.\ i\le j<k\Rightarrow \langle j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P$$

$$\langle i,n,M,\mathcal{P}\rangle \models_{\mathcal{C}'} P\ \textbf{proj}\ Q \quad \text{iff} \quad i=n\ \vee\ \exists M'\subseteq M.\ M'=\{k_0,k_1,\ldots,k_m\}\wedge i=k_0<k_1<\ldots<k_m=n\wedge \forall j.\ 0\le j<m\Rightarrow \langle k_j,k_{j+1},M,\mathcal{P}\rangle \models_{\mathcal{C}'} P \wedge \langle i,n,M',\mathcal{P}\rangle \models_{\mathcal{C}'} Q$$

where predicate $cons(S,a,b),\ S\in\mathbb{FN},\ a,b\in\mathbb{N}$ is defined as follows:

$$cons(S,a,b)\ \equiv a,b\in S\wedge a<b\wedge\neg\exists c\in C.\ a<c<b$$

Satisfaction of *PITL*^{1} formulas over $\mathcal{C}'$-structures can be shown to be necessary and sufficient w.r.t. satisfaction over intervals, for both describe exactly the same models (see theorem 14 in appendix B). Formally,

For any $\sigma\in\mathcal{I}$ and for any PITL^1 formula F, $\Theta'(\sigma)\models_{\mathcal{C}'} F$ iff $\sigma\models F$

It is worth saying that $\mathcal{C}'$-structures interpret any formula over a set M, however this is only compulsory for projected subformulas. For example, in P **proj** Q only subformula Q is required to be interpreted over sparse sets, and so P should be interpreted under a simpler semantics (as it translates itself to a less complex WS1S formula). The translation algorithm will then apply the proper (and simplest) interpretation to *PITL*^{1} subformulas in every recursive step, therefore returning the simplest WS1S formula possible (according to this strategy).

5. Translating PITL to MONA

A translation algorithm is obtained by just applying, for every different PITL construct, the semantic mapping we have derived in the previous section. First we will present the resulting MONA specifications obtained from the simplest semantics ($\mathcal{C}$-structures) and then we will show the necessary changes to handle *projection*. Let F be a *PITL*^{0} formula with atomic propositions $v_1, \ldots, v_m$. Then, the translation will produce a MONA specification (WS1S formula) P_F with $V_1, \ldots, V_m$ as corresponding free second-order variables, plus a first-order variable `n` to encode the length of the resulting interval (i.e. intervals which will be represent either models or counterexamples of F). The program P_F can be thought of as the result of the following steps (which is just an outline of our actual implementation). We use $output(S)$ to denote the assertion of a MONA statement S in the resulting program file.

1. *output*(`var1 n; var2` $V_1, \ldots, V_m$`;`);
2. *output*(`pred cons(var2 S, var1 a,b)= a in S & b in S & a<b &`
   $\sim$ `ex1 c:c in S & a<c & c<b;`);
3. *output*(*toWS1S*$(0, \mathtt{n}, F)$)

Table 1 defines the function *toWS1S*(i, n, F) which translates a *PITL*^{0} formula F to its corresponding MONA formula.

Table 1. *Translating PITL*^{0} *to MONA*

F	*toWS1S*(i, n, F)
v_j	i `<=` n `&` i `in` V_j
False	`false`
$\neg P$	$\sim$ *toWS1S*(i, n, P)
$\bigcirc P$	i `<` n `&` *toWS1S*$(i+1, n, P)$
$P \vee Q$	*toWS1S*(i, n, P) `|` *toWS1S*(i, n, Q)
empty	i `=` n
P **;** Q	`ex1 k:k >=` i `& k <=` n `&` *toWS1S*$(i, \mathtt{k}, P)$ `&` *toWS1S*$(\mathtt{k}, n, Q)$
P^*	`ex2 K:max K =` n `& min K =` i `&` `all1 k1,k2:cons(K,k1,k2) =>` *toWS1S*$(\mathtt{k1}, \mathtt{k2}, P)$
P **until** Q	`ex1 k:(k >=` i `& k <=` n `&` *toWS1S*$(\mathtt{k}, n, Q)$ `&` `all1 j:(j >=` i `& j < k =>` *toWS1S*$(\mathtt{j}, n, P)$`))`

Arguments i and n are meant to hold the MONA expressions denoting the current interval state and the current interval length, respectively. In other words, they correspond to indices i and n in $\langle i, n, \mathcal{P} \rangle$. The resemblance with the semantic definitions can be easily observed, as illustrated by the following example.

EXAMPLE 7. — A derivation of the WS1S/MONA formula corresponding to the *PITL*^{1} formula $\bigcirc A$ **;** $\bigcirc B$ is shown below:

$\rightarrow$ *toWS1S*$(0, \mathtt{n}, \bigcirc A$ `;` $\bigcirc B)$
$\rightarrow$ `ex1 k:k>=0 & k<=n &` *toWS1S*$(0, \mathtt{k}, \bigcirc A)$ `&` *toWS1S*$(\mathtt{k}, \mathtt{n}, \bigcirc B)$
$\rightarrow$ `ex1 k:k>=0 & k<=n & 0<k &` *toWS1S*$(1, \mathtt{k}, A)$ `&`
`k<n &` *toWS1S*$(\mathtt{k}+1, \mathtt{n}, B)$
$\rightarrow$ `ex1 k:k>=0 & k<=n & 0<k & 1 in A & k<n & k+1 in B`

Then, the MONA specification corresponding to $\bigcirc A$ `;` $\bigcirc B$ is as follows:

```
var1 n;
var2 A,B;
pred cons(var2 S, var1 a,b) = a in S & b in S & a<b &
                              ~ ex1 c:c in S & a<c & c<b;
ex1 k:k>=0 & k<=n & 0<k & 1 in A & k<n & k+1 in B;
```

□

In order to handle *projection*, and consequently to represent a $\mathcal{C}'$-structure, we add a new argument to *toWS1S*() to model the set of natural numbers M in $\langle i, n, M, \mathcal{P}\rangle$. The argument M will not be declared as a free second-order variable in the program, but rather it will appear existentially quantified when a formula under the scope of *projection* is translated. Consequently, the translation function itself is modified according to the definitions given on $\mathcal{C}'$-structures. Table 2 shows the translation of *chop* and *projection*, the translation of other operators in *PITL*^{1} are obtained similarly.

Table 2. *Translating PITL*^{1} *to MONA: chop and projection*

F	*toWS1S*(i, n, M, F)
P `;` Q	`ex1 k:k in` M `& k >=` i `& k <=` n `&` *toWS1S*$(i, \mathtt{k}, M, P)$ `&` *toWS1S*$(\mathtt{k}, n, M, Q)$
P **proj** Q	`ex2 K:K sub` M `& min K =` i `& max K =` n `&` `(all1 k1,k2:cons(K,k1,k2) =>` *toWS1S*$(\mathtt{k1}, \mathtt{k2}, M, P))$ `&` *toWS1S*$(i, n, \mathtt{K}, Q)$

6. PITL with past operators

This section shows how past operators such as *chop in the past*, *since* and *previous* can be encoded in WS1S. These operators are convenient for expressing properties about past computations, making PITL a language where a wider class of statements can be naturally described and verified. These operators were first proposed by Duan to extend ITL [DUA 96]. An axiomatisation and examples of the application of these operators in multimedia are also given in [BOW 03a]. In particular, [BOW 03a] presents *chop in the past* as a primitive operator from which other past operators (e.g. *since* and *previous*) can be derived (although the logic they use, *MEXITL*, handles predicates and therefore is more expressive than PITL).

The interpretation of past operators in PITL requires not only the subinterval under consideration (as is sufficient for future operators), but also the sequence of states which have been passed through. Therefore, the inclusion of past operators in the language forces the semantics of all operators to be re-defined in terms of a pair (σ, j), i.e. the original interval and the current state. State j, then, divides σ into a current subinterval $(\sigma)^j$ and a past history $[\sigma]^j$, both now accessible in the operator definitions. Next we present the interpretation of past operators in ITL (this is illustrated in Fig. 8). The interpretation of future operators on (σ, j) (shown in appendix A) can be easily obtained from the ones given in section 2.

$$(\sigma, j) \models P \stackrel{\sim}{;} Q \quad \text{iff} \quad \exists k \in \mathbb{N}.\, 0 \leq k \leq j \ \wedge\ (\sigma, k) \models P \wedge ((\sigma)^k, j-k) \models Q$$

$$(\sigma, j) \models P\, \mathcal{S}\, Q \quad \text{iff} \quad \exists k \in \mathbb{N}.\, 0 \leq k \leq j \ \wedge\ (\sigma, k) \models Q \wedge \forall r \in \mathbb{N}.\, k < r \leq j \Rightarrow (\sigma, r) \models P$$

$$(\sigma, j) \models \ominus P \quad \text{iff} \quad j > 0 \ \wedge\ (\sigma, j-1) \models P$$

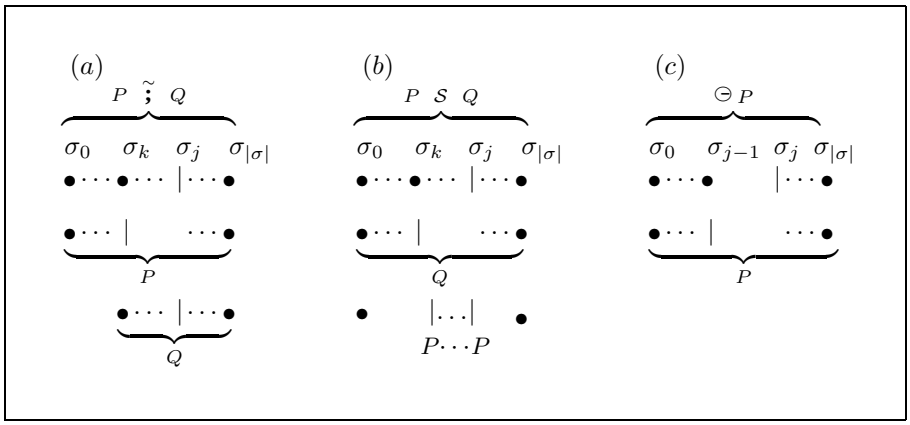

Figure 8. *Past operators: chop in the past (a), since (b) and previous (c)*

$P \stackrel{\sim}{;} Q$ (Fig. 8a) is satisfied by an interval such that 1) P holds over the larger interval resulting from moving the start of the interval (denoted by |) into the past, and 2) Q holds over the original interval according to a past history that is truncated at the start of the interval over which P holds. $P\, \mathcal{S}\, Q$ (Fig. 8b) is satisfied by an interval such that either 1) Q holds over the current interval or 2) Q holds over a larger interval resulting from moving the start of the interval into the past and P holds over every suffix of this larger interval which starts up to the current state. Finally, $\ominus P$ (Fig. 8c) is satisfied by an interval such that P holds over a larger interval resulting from moving the start of the interval one state into the past.

Consequently with (σ, j), the interpretation of past operators in WS1S requires $\mathcal{C}'$-structures to include a new parameter.

DEFINITION 8. — *A $\mathcal{C}''$-structure is a tuple $\langle i, j, n, M, \mathcal{P}\rangle$ where $M \in \mathbb{FN}$, $i, j, n \in M$, $i \leq j \leq n$, and $\mathcal{P} = \{(V_1, S_1), \ldots, (V_m, S_m)\}$ s.t. $S_k \in \mathbb{FN}$, $1 \leq k \leq m$.*

A $\mathcal{C}''$-structure $\langle i, j, n, M, \mathcal{P}\rangle$ is then intended to model the sparse subinterval $(\sigma_{t_0}, \ldots, \sigma_{t_s}, j)$, where $i = t_0 < t_1 < \ldots < t_s = n$, $\{t_0, t_1, \ldots, t_s\} = M \cap \{i, \ldots, n\}$. Let $\mathcal{C}''$ be the set of all $\mathcal{C}''$-structures. The mapping between intervals and $\mathcal{C}''$-structures, $\Theta'' : \mathcal{I} \rightarrow \mathcal{C}''$ is defined as follows:

DEFINITION 9. — *$\Theta''(\sigma) = \langle 0, 0, |\sigma|, M, \mathcal{P}\rangle$ where $\forall i.\ i \in M \Leftrightarrow 0 \leq i \leq |\sigma|$, and $\mathcal{P} = \{(V_j, Set(v_j, \sigma)) \mid v_j \in \mathcal{A},\ V_j \in \mathcal{V},\ 1 \leq j \leq m\}$.*

The satisfaction relation $\models_{\mathcal{C}''}$ for past operators is shown below. Since future operators do not depend on past states, the corresponding satisfaction relation is just a straightforward generalisation of $\models_{\mathcal{C}'}$ (presented in section 4). Thus, as an example amongst the future operators, we just show the relation for *chop* (see appendix A for definitions of all future operators over intervals with past history). The set M, inherited from $\mathcal{C}'$-structures to handle the translation of formulas under the scope of *projection*, is also necessary here because the past history can also be *sparse* as a consequence of past operators under *projection*.

$$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P\ ;\ Q \quad \text{iff} \quad \exists k \in M.\ j \leq k \leq n \wedge \langle i, j, k, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge \langle i, k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} Q$$

$$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P\ \widetilde{;}\ Q \quad \text{iff} \quad \exists k \in M.\ i \leq k \leq j \wedge \langle i, k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge \langle k, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} Q$$

$$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P\ \mathcal{S}\ Q \quad \text{iff} \quad \exists k \in M.\ i \leq k \leq j \wedge \langle i, k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} Q \wedge \forall r \in M.\ k < r \leq j \Rightarrow \langle i, r, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P$$

$$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} \ominus P \quad \text{iff} \quad cons(M, k, j)\ \wedge\ \langle i, k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P$$

It can be shown that this interpretation is correct in the following sense (see Theorem 16 in appendix B):

For any $\sigma \in \mathcal{I}$ and for any PITL formula F, $\Theta''(\sigma) \models_{\mathcal{C}''} F$ iff $(\sigma, 0) \models F$

Notice that the translation function *toWS1S*() now includes a new parameter j, initially set to 0 (i.e. the current state is the first state of the interval, $i = j = 0$). Effectively, then, the translation function encodes the three indices i, j, n used in $\mathcal{C}''$-structures. Again, and as previously mentioned in Section 5 for the translation of *PITL*[1], neither j nor M will be declared as free variables in the resulting MONA specification; they will appear instead as quantified variables. Table 3 shows the translation of past operators.

Table 3. *Translating PITL to MONA: past operators*

F	*toWS1S*(i, j, n, M, F)
$P \mathbin{\widetilde{;}} Q$	`ex1 k:k in` M `& k >=` i `& k <=` j `&` *toWS1S*$(i, \mathtt{k}, n, M, P)$ `&` *toWS1S*$(\mathtt{k}, j, n, M, Q)$
$P \mathcal{S} Q$	`ex1 k:k in` M `& `i `<= k <=` j `&` *toWS1S*$(i, \mathtt{k}, n, M, Q)$ `&` `all1 r:r in` M `& k < r <=` j `=>` *toWS1S*$(i, \mathtt{r}, n, M, P)$
$\ominus P$	`cons(`M`,k,`j`) &` *toWS1S*$(i, \mathtt{k}, n, M, P)$

We have sketched an algorithm which translates a given PITL formula F into an equivalent MONA specification P_F (we have implemented this algorithm in our tool PITL2MONA, described in Section 8). We will now define this equivalence in formal terms. Let W_F be the WS1S formula represented by the MONA specification P_F. We can relate the models of W_F (as valuations for the free variables in W_F) and the models of F (as PITL intervals) as follows. Let $(n, S_1, \ldots, S_m) \models_{WS1S} W_F$ denote a model for W_F, where $n \in \mathbb{N}$ and $S_1, \ldots, S_m$, $S_i \in \mathbb{FN}, 1 \leq i \leq m$, denote a valuation for the free variables `n`, $V_1, \ldots, V_m$ in W_F. Then the following holds:

$(n, S_1, \ldots, S_m) \models_{WS1S} W_F$ iff
$\exists \sigma \in \mathcal{I}.\ |\sigma| = n \ \wedge\ \sigma \models F \ \wedge\ \forall \sigma'.\ (\sigma' \models F \Rightarrow n \leq |\sigma'|) \wedge$
$\Theta''(\sigma) = \langle 0, 0, n, M, \mathcal{P} \rangle$,
where $\forall i.\ i \in M \Leftrightarrow 0 \leq i \leq n$ and $\mathcal{P} = \{(V_1, S_1), \ldots, (V_m, S_m)\}$

In other words, when MONA analyses W_F it produces an instantiation for its free variables which represents the shortest model for F. Counterexamples for W_F and F can be related in a similar way:

$(n, S_1, \ldots, S_m) \not\models_{WS1S} W_F$ iff
$\exists \sigma \in \mathcal{I}.\ |\sigma| = n \ \wedge\ \sigma \not\models F \ \wedge\ \forall \sigma'.\ (\sigma' \not\models F \Rightarrow n \leq |\sigma'|) \wedge$
$\Theta''(\sigma) = \langle 0, 0, n, M, \mathcal{P} \rangle$,
where $\forall i.\ i \in M \Leftrightarrow 0 \leq i \leq n$ and $\mathcal{P} = \{(V_1, S_1), \ldots, (V_m, S_m)\}$

We will not offer a proof of these claims in this paper. However, it is not hard to show that their correctness directly relies on the correctness of the semantic translation from PITL to WS1S formulas (Theorem 16, appendix B), and the fact that MONA returns the shortest model (counterexample) for the analysed WS1S formula (as explained in Section 3).

7. Examples

This section illustrates the use of PITL and its decision procedure in some canonical examples, i.e. they represent commonly-found problems in verification of practical

applications. Also, this will give the reader an idea of the strengths and limitations of the logic and the implementation of its decision procedure in MONA.

7.1. *2-Dining Philosophers*

Here we show a PITL formalisation for this well-known multiple-resource allocation problem. Every philosopher alternates between a thinking phase and a phase in which he becomes hungry and wishes to eat. We consider only two philosophers, Aristotle and Plato which are sharing a single set of cutlery, i.e. one fork and one knife. Nevertheless, there is no intrinsic reason why the specification could not be enlarged to more philosophers. Formulas describe the behaviour of philosophers as the set of possible allocations of fork and knife, and the moments when they are not thinking, i.e. when philosophers are either trying to pick up the cutlery (they are holding only one piece of cutlery and are waiting for the other to start eating) or eating (when philosophers are holding both the fork and the knife). Propositions *not_thinka* and *not_thinkp* denote that Aristotle, respectively Plato, are either trying to eat (trying to pick up both the fork and the knife), or already eating. Proposition *af* (*ak*) denotes that Aristotle is currently holding the fork (knife). Similarly, *pf* and *pk* denote the same property for Plato. We use operators $\wedge$, $\Rightarrow$, $\Diamond$ and $\Box$ as defined in section 2.

1) From time to time, philosophers stop thinking:

$$F_1 \equiv \Box\Diamond not\_thinka \wedge \Box\Diamond not\_thinkp$$

Because of PITL's finite models, this formula implies that both philosophers will not be thinking at the end of every model.

2) When philosophers are not thinking, they will try to pick up the cutlery:

$$F_2 \equiv \Box(not\_thinka \vee not\_thinkp \Rightarrow (af \mid pf) \wedge (ak \mid pk))$$

Because of formula F_1, this formula implies that no piece of cutlery will remain on the table at the end of every model.

3) Philosophers will not attempt to pick up the cutlery if they are thinking:

$$F_3 \equiv \Box(\neg not\_thinka \Rightarrow \neg(af \mid ak)) \wedge \Box(\neg not\_thinkp \Rightarrow \neg(pf \mid pk))$$

4) Philosophers cannot hold the cutlery at the same time:

$$F_4 \equiv \Box\neg(af \wedge pf) \wedge \Box\neg(ak \wedge pk)$$

5) Philosophers do not release the fork (knife) if they do not have also the knife (fork):

$$\begin{aligned} F_5 \equiv \quad & \Box(\neg\textbf{empty} \wedge af \wedge \neg ak \Rightarrow \bigcirc af) \wedge \Box(\neg\textbf{empty} \wedge ak \wedge \neg af \Rightarrow \bigcirc ak) \wedge \\ & \Box(\neg\textbf{empty} \wedge pf \wedge \neg pk \Rightarrow \bigcirc pf) \wedge \Box(\neg\textbf{empty} \wedge pk \wedge \neg pf \Rightarrow \bigcirc pk) \end{aligned}$$

Notice the use of $\neg$**empty** in the implications to prevent the description of infinite models, which results in unsatisfiable PITL formulas.

6) If a philosopher is neither thinking nor he has both the fork and the knife, then he remains in the "not thinking" state:

$$\begin{aligned} F_6 \equiv \quad & \Box(\neg\textbf{empty} \wedge not\_thinka \wedge \neg(af \wedge ak) \Rightarrow \bigcirc not\_thinka) \wedge \\ & \Box(\neg\textbf{empty} \wedge not\_thinkp \wedge \neg(pf \wedge pk) \Rightarrow \bigcirc not\_thinkp) \end{aligned}$$

7) Philosophers release both the fork and the knife at the same time:

$$\begin{aligned} F_7 \equiv \quad & \Box(af \wedge ak \wedge \bigcirc\neg(af \wedge ak) \Rightarrow \bigcirc\neg(af \mid ak)) \wedge \\ & \Box(pf \wedge pk \wedge \bigcirc\neg(pf \wedge pk) \Rightarrow \bigcirc\neg(pf \mid pk)) \end{aligned}$$

Let *Sys* denote the conjunction of the previous formulas (F_1 to F_7). We can use the decision procedure to verify a number of properties. For example, the property "*At least one philosopher eventually eats*" can be expressed as:

$$\begin{aligned} AtLeastOneEats \equiv \quad & \Box(\neg\textbf{empty} \wedge (not\_thinka \mid not\_thinkp) \Rightarrow \\ & \Diamond(af \wedge ak \mid pf \wedge pk)) \end{aligned}$$

Similarly, the property "*Both philosophers will eventually eat*" can be described by

$$\begin{aligned} EveryoneEats \equiv \quad & \Box(\neg\textbf{empty} \wedge not\_thinka \Rightarrow \Diamond(af \ \wedge \ ak)) \wedge \\ & \Box(\neg\textbf{empty} \wedge not\_thinkp \Rightarrow \Diamond(pf \ \wedge \ pk)) \end{aligned}$$

These properties can be verified in a number of general contexts:

– Satisfiability of $Sys \wedge AtLeastOneEats$ ensures that there is at least an "acceptable" run in the system, i.e. a system execution where at least one philosopher eats. Notice that if we check this formula for validity instead of satisfiability, then any interval which invalidates the formulas defining *Sys* will be presented by MONA as a counterexample. Therefore, typically, we just check satisfiability as a way to ensure

that our specification is at least consistent, i.e. that there exists at least an interval with some minimum correctness requirements (in our case the formula *AtLeastOneEats*) which satisfies the specification.

– Validity of $Sys \Rightarrow AtLeastOneEats$ ensures that all runs of the system make progress according to *AtLeastOneEats*, i.e. at least one philosopher eats in all possible system executions. Notice here, as well, that the satisfiability checking previously discussed becomes handy; it ensures that the implication is not "vacuously" valid due to an inconsistent antecedent (*Sys*).

– Validity of $Sys \Rightarrow EveryoneEats$ ensures that all runs of the system make progress according to *EveryoneEats*. If this is the case, we can consider the system to be *free from starvation*.

An inspection of these formulas reveals that philosophers are not precluded from holding a resource that the other philosopher needs. For example, it is possible for Aristotle to hold the fork (*af*) while Plato holds the knife (*ak*). Because they will not release the fork or the knife until they manage to get both, every philosopher remains blocked, requesting the fork (or the knife) the other philosopher is holding. Consequently properties *AtLeastOneEats* and *EveryoneEats* are not validated (i.e. there is a run of the system where these properties do not hold), and thus formulas $Sys \Rightarrow AtLeastOneEats$ and $Sys \Rightarrow EveryoneEats$ are not valid. The PITL decision procedure returns these undesired situations as MONA counterexamples. For example, MONA returns `af={} ak={1} pf={1} pk={}` as an interval where formula $Sys \Rightarrow AtLeastOneEats$ is not satisfiable. This is actually produced for the WS1S formula resulting from our translation algorithm. As explained before, our semantics assigns second-order variables to PITL propositions. Therefore the counterexample represents the interval $\sigma = \langle \{\}_0, \{ak, pk\}_1 \rangle$. In other words, the lack of response is a consequence of Aristotle holding the knife ($ak = \{1\}$) and Plato holding the fork ($pf = \{1\}$) at the same time, and of the fact that they cannot release them until they get both. We can prevent this situation from happening if the allocation of the fork and the knife is ordered, e.g. if the fork must always be picked up before the knife:

$$Sys1 \equiv Sys \wedge \Box(ak \Rightarrow af) \wedge \Box(pk \Rightarrow pf)$$

In other words, MONA will find that formula $Sys1 \Rightarrow AtLeastOneEats$ is valid. Notice, also, that because at the end of every model no piece of cutlery remains on the table (formula F_2), formula *Sys1* also implies that one philosopher will always be eating at the end of the model. This is a consequence of adapting a problem initially devised for infinite computations to the finite framework of PITL. *Sys1* is, however, not enough to prevent starvation as philosophers are never required to stop eating (thus preventing the other philosopher from doing so). Starvation can be prevented if philosophers are not allowed to eat forever (again, ¬**empty** copes with PITL's finite models):

$$Sys2 \equiv Sys1 \wedge \neg\Diamond(\neg\textbf{empty} \wedge \Box(af \wedge ak)) \wedge \neg\Diamond(\neg\textbf{empty} \wedge \Box(pf \wedge pk))$$

Again, MONA will confirm that formula $Sys2 \Rightarrow EveryoneEats$ is valid. We consider below a different solution to the dining-philosophers problem which illus-

trates the use of the logic in more prescriptive specifications. In this solution, behaviours are described as successful cycles, i.e. philosophers eventually get both the fork and the knife.

$$
\begin{array}{llll}
\mathit{eatreqa} & \equiv \neg af \wedge \neg ak \wedge \mathit{reqa} & \mathit{eatreqp} & \equiv \neg pk \wedge \neg pf \wedge \mathit{reqp} \\
\mathit{forka} & \equiv af \wedge \neg ak \wedge \neg \mathit{reqa} & \textit{knifea} & \equiv ak \wedge \neg af \wedge \neg \mathit{reqa} \\
\mathit{forkp} & \equiv pf \wedge \neg pk \wedge \neg \mathit{reqp} & \textit{knifep} & \equiv pk \wedge \neg pf \wedge \neg \mathit{reqp} \\
\mathit{eata} & \equiv af \wedge ak \wedge \neg \mathit{reqa} & \mathit{eatp} & \equiv pk \wedge pf \wedge \neg \mathit{reqp} \\
\mathit{thinka} & \equiv \neg af \wedge \neg ak \wedge \neg \mathit{reqa} & \mathit{thinkp} & \equiv \neg pf \wedge \neg pk \wedge \neg \mathit{reqp}
\end{array}
$$

$$
\begin{array}{ll}
\textit{Aristotle} & \equiv (\Box \mathit{thinka} \,;\, \bigcirc\Box \mathit{eatreqa} \,;\, \bigcirc(\mathit{forka} \mid \mathit{knifea}) \,;\, \bigcirc\Box \mathit{eata} \,;\bigcirc\Box \mathit{thinka})^* \\
\textit{Plato} & \equiv (\Box \mathit{thinkp} \,;\, \bigcirc\Box \mathit{eatreqp} \,;\, \bigcirc(\mathit{forkp} \mid \mathit{knifep}) \,;\, \bigcirc\Box \mathit{eatp} \,;\bigcirc\Box \mathit{thinkp})^*
\end{array}
$$

Here we are modelling every philosopher with three phases. Let us take as an example the behaviour for Aristotle. First he is thinking and not hungry (i.e. he is not requesting to eat) for an indeterminate period of time ($\Box\neg \mathit{reqa}$). Eventually, he will become hungry and will request to eat (*eatreqa*) until he manages to pick up either the fork or the knife ($\mathit{forka} \mid \mathit{knifea}$). Philosophers can pick up the cutlery in any order. Then he will eat for a period of time ($\bigcirc\Box \mathit{eata}$) until he starts thinking again, and the cycle is resumed. We can see that at the end (and start) of every cycle both philosophers are thinking and both pieces of cutlery are on the table. Let

$$
\begin{array}{ll}
\mathit{Sys3} & \equiv \quad \mathit{Aristotle} \wedge \mathit{Plato} \wedge (\Box\neg(af \wedge pf)) \wedge (\Box\neg(ak \wedge pk)) \\
\mathit{StarvationFree} & \equiv \quad \Box(\neg \mathbf{empty} \wedge \mathit{reqa} \Rightarrow \Diamond \mathit{eata}) \wedge \Box(\neg \mathbf{empty} \wedge \mathit{reqp} \Rightarrow \Diamond \mathit{eatp})
\end{array}
$$

The decision procedure can be used to prove that formula $\mathit{Sys3} \Rightarrow \mathit{StarvationFree}$ is valid, and so whenever each philosopher requests a resource he eventually obtains it. This is not surprising since the behaviour of philosophers was encoded as successive iterations of the *star* operator (formulas *Aristotle* and *Plato*), which includes formulas *eata* and *eatp* in the specification. Therefore, consistency of the conjunction $\mathit{Aristotle} \wedge \mathit{Plato}$ ensures that both philosophers eventually eat in every cycle.

We believe that the dining philosophers example suffices to illustrate the capabilities of PITL as a property-specification language, and the applicability of its decision procedure in the validation of specifications. For example, the decision procedure may check satisfiability of a property expressing a requirement for a safety-critical system. If satisfiability is not ensured then further verification stages will be based on ill-formed requirements, which may dangerously give unjustified confidence in the system.

7.2. *The Beethoven problem*

This problem has been used in Multimedia to show the capabilities of modelling languages (see e.g. [BOW 03a]). The Beethoven problem we are considering here (since we are working on a propositional logic, this is necessarily a simplified version of the predicate-based description given in [BOW 03a]) consists of the description of two synchronous multimedia streams; an audio stream which comprises the four movements of Beethoven's Fifth Symphony, and a video stream which displays the title of every movement while the corresponding movement is playing. These requirements can be stated as follows:

1) Play the four movements of the symphony in sequence with a gap of 5 seconds between each movement.

2) During each movement display an appropriate title for 2 seconds. Each title is a still image. Repeat the 2-second display of each title every 4 seconds during the corresponding movement.

3) The presentation will then result from composing these two streams in parallel. We want to determine whether slowing down each stream before composition is equivalent to slowing down the composite stream. Here we represent the slowing down of a multimedia presentation by placing delays between consecutive states in the original model. Fig. 9 illustrates the concept, and its natural PITL interpretation using the *projection* operator: here P denotes the PITL formula which represents the presentation, and the delay was set to 2 sec.

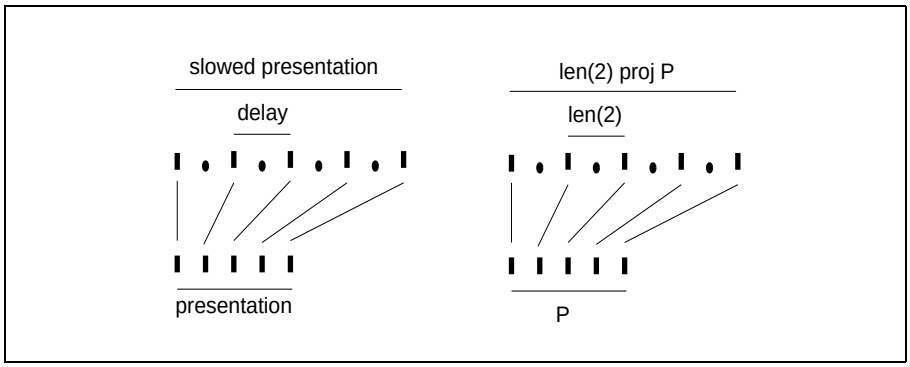

Figure 9. *Slowing down a media item*

We will represent each movement m_i as a pair of bounding propositions and a fixed delay (10 sec.) in between, i.e. propositions $start_i$ and end_i will denote the beginning and end of movement m_i. The title for movement m_i will also be represented as a proposition, t_i, which will be true every time the corresponding image is displayed. Projection will be used to represent the slowing down of a stream (we have chosen a delay of 2 sec.) The following PITL formulas describe a possible solution to

the Beethoven problem:

1) The four movements:

$$
\begin{array}{rcl}
m_1 & \equiv & (start_1 \wedge \textbf{empty})\ \textbf{;}\ \textbf{len}(10)\ \textbf{;}\ (end_1 \wedge \textbf{empty}) \\
m_2 & \equiv & (start_2 \wedge \textbf{empty})\ \textbf{;}\ \textbf{len}(10)\ \textbf{;}\ (end_2 \wedge \textbf{empty}) \\
m_3 & \equiv & (start_3 \wedge \textbf{empty})\ \textbf{;}\ \textbf{len}(10)\ \textbf{;}\ (end_3 \wedge \textbf{empty}) \\
m_4 & \equiv & (start_4 \wedge \textbf{empty})\ \textbf{;}\ \textbf{len}(10)\ \textbf{;}\ (end_4 \wedge \textbf{empty})
\end{array}
$$

2) Movement-bounding actions only occur once:

$$
\begin{array}{rl}
F_1 \equiv & (\neg start_1\ \textbf{until}\ start_1)\ \textbf{;}\ \bigcirc\Box\neg start_1 \wedge \\
& (\neg start_2\ \textbf{until}\ start_2)\ \textbf{;}\ \bigcirc\Box\neg start_2 \wedge \\
& (\neg start_3\ \textbf{until}\ start_3)\ \textbf{;}\ \bigcirc\Box\neg start_3 \wedge \\
& (\neg start_4\ \textbf{until}\ start_4)\ \textbf{;}\ \bigcirc\Box\neg start_4 \wedge \\
& (\neg end_1\ \textbf{until}\ end_1)\ \textbf{;}\ \bigcirc\Box\neg end_1 \wedge \\
& (\neg end_2\ \textbf{until}\ end_2)\ \textbf{;}\ \bigcirc\Box\neg end_2 \wedge \\
& (\neg end_3\ \textbf{until}\ end_3)\ \textbf{;}\ \bigcirc\Box\neg end_3 \wedge \\
& (\neg end_4\ \textbf{until}\ end_4)\ \textbf{;}\ \bigcirc\Box\neg end_4
\end{array}
$$

3) The audio stream:

$$A \equiv m_1\ \textbf{;}\ \textbf{len}(5)\ \textbf{;}\ m_2\ \textbf{;}\ \textbf{len}(5)\ \textbf{;}\ m_3\ \textbf{;}\ \textbf{len}(5)\ \textbf{;}\ m_4\ \textbf{;}\ \textbf{True}$$

Notice that F_1 ensures that actions $start_i$ and end_i are *framed* in A, i.e. they will only occur when the corresponding movement is placed in the specification of the stream.

4) Displaying a title for 2 sec., once every 2 sec.:

$$
\begin{array}{rcl}
dt_1 & \equiv & (((t_1 \wedge \textbf{len}(1))^* \wedge \textbf{len}(2))\ \textbf{;}\ \textbf{len}(2))^* \\
dt_2 & \equiv & (((t_2 \wedge \textbf{len}(1))^* \wedge \textbf{len}(2))\ \textbf{;}\ \textbf{len}(2))^* \\
dt_3 & \equiv & (((t_3 \wedge \textbf{len}(1))^* \wedge \textbf{len}(2))\ \textbf{;}\ \textbf{len}(2))^* \\
dt_4 & \equiv & (((t_4 \wedge \textbf{len}(1))^* \wedge \textbf{len}(2))\ \textbf{;}\ \textbf{len}(2))^*
\end{array}
$$

5) Titles can only be displayed during the corresponding movement:

$$
\begin{array}{rl}
F_2 \equiv & (\neg t_1\ \textbf{until}\ start_1) \wedge \Diamond(end_1\textbf{;}\ \bigcirc\Box\neg t_1) \wedge \\
& (\neg t_2\ \textbf{until}\ start_2) \wedge \Diamond(end_2\textbf{;}\ \bigcirc\Box\neg t_2) \wedge \\
& (\neg t_3\ \textbf{until}\ start_3) \wedge \Diamond(end_3\textbf{;}\ \bigcirc\Box\neg t_3) \wedge \\
& (\neg t_4\ \textbf{until}\ start_4) \wedge \Diamond(end_4\textbf{;}\ \bigcirc\Box\neg t_4)
\end{array}
$$

Formula F_2 ensures that t_i is *framed* in A, i.e. the display of a title during the corresponding movement.

6) Synchronising a title with the corresponding movement:

$$
\begin{array}{rcl}
synct_1 & \equiv & \textbf{halt}(start_1)\ \textbf{;}\ ((dt_1\ \textbf{;}\ \bigcirc\neg((\Box\neg t_1) \wedge (\textbf{True}\ \textbf{;}\ \textbf{len}(2)))) \wedge \textbf{halt}(end_1)) \\
synct_2 & \equiv & \textbf{halt}(start_2)\ \textbf{;}\ ((dt_2\ \textbf{;}\ \bigcirc\neg((\Box\neg t_2) \wedge (\textbf{True}\ \textbf{;}\ \textbf{len}(2)))) \wedge \textbf{halt}(end_2)) \\
synct_3 & \equiv & \textbf{halt}(start_3)\ \textbf{;}\ ((dt_3\ \textbf{;}\ \bigcirc\neg((\Box\neg t_3) \wedge (\textbf{True}\ \textbf{;}\ \textbf{len}(2)))) \wedge \textbf{halt}(end_3)) \\
synct_4 & \equiv & \textbf{halt}(start_4)\ \textbf{;}\ ((dt_4\ \textbf{;}\ \bigcirc\neg((\Box\neg t_4) \wedge (\textbf{True}\ \textbf{;}\ \textbf{len}(2)))) \wedge \textbf{halt}(end_4))
\end{array}
$$

Here, formula $\neg((\Box\neg t_i) \wedge (\textbf{True}\ \textbf{;}\ \textbf{len}(2)))$ ensures that the title is displayed during the whole movement, i.e. it forces a maximal number of iterations for the *star* operator applied in dt_i.

7) The video stream:

$$V \equiv synct_1\ \textbf{;}\ \textbf{True}\ \textbf{;}\ synct_2\ \textbf{;}\ \textbf{True}\ \textbf{;}\ synct_3\ \textbf{;}\ \textbf{True}\ \textbf{;}\ synct_4\ \textbf{;}\ \textbf{True}$$

It is interesting to see how the conjunction $A \wedge V$ effectively models the composition of the audio and video streams; this is done by virtue of actions $start_i$ and end_i, which appear both in A (defining the movement m_i) and in V (synchronising the period when titles are displayed in F_2 and $synct_i$). The decision procedure can be used to prove that slowing down the individual streams (A and V) before composition results in a system which is equivalent to slowing down the composed stream ($A \wedge V$). This property can be expressed as follows, where we can see how the *projection* operator becomes handy in such specifications.

$$P \equiv (\textbf{len}(2)\ \textbf{proj}\ A) \wedge (\textbf{len}(2)\ \textbf{proj}\ V) \Leftrightarrow \textbf{len}(2)\ \textbf{proj}\ (A \wedge V)$$

MONA, then, checks the validity of formula $F_1 \wedge F_2 \Rightarrow P$. As an example application of past operators, we can verify that no title is shown before the corresponding movement begins:

$$
\begin{array}{rcl}
P & \equiv & \Box(t_1 \Rightarrow \diamondminus start_1) \wedge \Box(t_2 \Rightarrow \diamondminus start_2) \wedge \\
 & & \Box(t_3 \Rightarrow \diamondminus start_3) \wedge \Box(t_4 \Rightarrow \diamondminus start_4)
\end{array}
$$

where the $\diamondminus$ past operator is defined as $\diamondminus F \equiv F\ \tilde{;}\ \textbf{True}$, and the formula to check is $F_1 \wedge F_2 \wedge A \wedge V \Rightarrow P$. We have used our implementation of the decision procedure in MONA to check that all formulas to verify are indeed valid.

8. PITL2MONA and experimental results

This section briefly presents some implementation details about PITL2MONA, and provides a comparison between PITL2MONA/MONA and LITE (Kono's tableau-based implementation). PITL2MONA is built as a front-end which takes a PITL formula and returns the corresponding MONA specification. Input and output are handled just as ASCII files. Basically, the tool is built as a parser of PITL syntax which translates the formula structure into a formula written in the MONA input language (WS1S plus syntactic sugar). The parser is automatically generated using a LEX/YACC toolkit. Translation code is written in C and embedded in the YACC grammar specification files; this code implements the algorithm sketched by the function *toWS1S()* (as described in Sections 5 and 6).

Table 4 shows the performance of both tools for a number of parameterised problems. Only execution time is shown; no memory consumption is output by LITE, so we cannot compare that with the amount of memory required by MONA. Also, although LITE does show the size of the tableau (in terms of number of nodes, subterms and transitions), structural differences prevent us from comparing these figures with the size of the corresponding MONA automaton/BDD. Table 5 shows a characterisation of the complexity of the PITL specifications in terms of

1) Length of the formula (in KB) and maximum amount of memory consumed by MONA to analyse the formula (in MB). This memory requirement corresponds to the biggest automaton generated by MONA during the analysis of the WS1S source formula.

2) Number of free variables. Free variables in the PITL formula result in a similar number of free variables in the corresponding WS1S formula, which in turn determines the size of the automaton alphabet and its transition function. A BDD-based implementation of a transition function usually leads to exponential compression, but depending on the WS1S formula this compression may not be achieved, and so in the worst case the representation will still be exponential in the number of variables. Also, the order in which variables are declared in the MONA specification can adversely influence the size of the BDD, and the problem of deciding an optimal ordering is known to be NP-complete [KLA 01, BRY 86].

3) Maximum nesting depth of quantifiers. The nesting of quantifiers of the same kind (i.e. either existential or universal quantifiers) does not necessarily relate to a non-elementary complexity, as the resulting automaton can be obtained by applying a combined automata-projection operation (Section 3). Nevertheless the translation of quantifiers requires automata determinisation, which can result in an exponential number of states.

4) Maximum nesting depth of alternating quantifiers. As mentioned in Section 3, the time and space for translating WS1S formulas to automata is, in the worst case, bounded from below by a stack of exponentials whose height is proportional to the depth of quantifier alternation. This measure will give us, therefore, a good idea about how hard to solve specifications are.

The n-Dining philosophers problem is parameterised in the number of philosophers; we have tested a formalisation which is similar to the second specification appearing in section 7. The n-movement Beethoven problem is parameterised by the number of movements; we have tested the formalisation given in section 7 (without the past operators, which are not supported by LITE). The n-floor Lift controller problem is parameterised by the number of floors the lift serves; the formalisation we have tested has been adapted from one proposed by Pandya for his tool DCVALID. Experiments were conducted with a Sun Solaris 5.9 running on a Sun v480 station, with 2x900MHZ US3 processors, and 4GB RAM. SICStus Prolog v. 3.10.1. was used to run LITE.

Table 4. *PITL2MONA/MONA vs. LITE: approx. execution time (in secs.)*

Example	n	*PITL2MONA/MONA*	*LITE*
n-Dining philosophers	5	2	163
	7	52	-
	10	-	-
n-movement Beethoven	4	4	7339
	8	120	-
	10	-	-
n-floor Lift controller	5	3	-
	10	110	-

Table 5. *PITL2MONA/MONA, the complexity of specifications: length, max. memory requirements, number of free variables, max. nesting depth of general quantifiers ($\exists$) and max. nesting depth of alternating quantifiers ($\exists\forall$).*

Example	n	*len. (KB)*	*mem. (MB)*	*# free vars.*	$\exists$	$\exists\forall$
n-Dining philosophers	5	6	2	11	6	4
	7	11	309	15	6	4
	10	15	-	21	6	4
n-movement Beethoven	4	69	3	13	27	3
	8	209	177	25	43	3
	10	298	-	31	51	3
n-floor Lift controller	5	16	1	28	4	3
	10	36	87	53	4	3

Table 4 suggests that MONA runs considerably faster than LITE, which is tableau-based. Of all specifications tested, only those corresponding to 10-Dining philosophers and 10-movement Beethoven problems could not be handled by MONA. Both the number of variables and the kind of operators involved in the source formula, e.g. *chop*, *star*, *projection* and negation, resulted in WS1S formulas with heavy use of quantifiers and quantifier alternation, i.e. WS1S formulas which are prone to exhibit the worst-case, non-elementary complexity of the decision procedure. This complexity reveals itself in the form of a state-explosion problem, with BDDs which are too

large for MONA to represent (even though MONA can handle BDDs with more than 16 million nodes [KLA 01]). The impact of variable ordering and nesting-depth of quantifiers is confirmed by Table 5. For example, the 7-Dining philosophers problem has roughly the same number of variables as the 4-movement Beethoven problem, and is much shorter. However the former is much harder to solve both in time and space; notice that the MONA specification for the 7-Dining philosophers problem has more variables (15 against 13), possibly a worse variable ordering [2] and a deeper nesting of alternating quantifiers (4 against 3). Moreover, and since the maximum nesting depths of alternating quantifiers do not vary w.r.t. the parameter size, the table suggests that the number of variables and the particular variable orderings represent the main cause for the performance degradation of these examples.

But the performance of LITE was even worse; for example, MONA was almost 2000 times faster than LITE to verify the 4-movement Beethoven problem. Moreover, most of the problems could not be verified by LITE at all. We therefore claim that even when both an automata-based and a tableau-based decision procedure for PITL will equally suffer of a worst-case, non-elementary complexity, a) the number of optimisations implemented in MONA and b) the expansion rules, generation of normal forms and subterms required by the tableau method make the automata-based decision procedure far more efficient.

Discussion.

A brief analysis on the translation from PITL to WS1S formulas (Sections 4 and 6) easily reveals those PITL formulas which may exhibit the logic's non-elementary complexity. In fact, these formulas must include a combination of operators which eventually result in WS1S formulas with alternating quantifiers. The following are some examples of such combinations; a) alternated nesting of *chop* (*chop in the past*) and negation, b) nesting of *star* (*until*, *since*, *projection*) and c) alternated nesting of *next* (*previous*) and negation under the scope of *projection*.

9. Conclusions

We have explored the connection between PITL and WS1S, and presented a decision procedure for PITL based on a translation to WS1S and the MONA tool. PITL includes *chop*, *star*, *projection*, *until* and past operators such as *chop in the past*, *since* and *previous*, so it is expressive enough to encode an interesting range of formulas. For example, temporal logic operators like $\Diamond$ and $\Box$ can be derived from the current set, and so formulas which are commonly used in verification can be expressed. We have developed a WS1S-based semantics for PITL, and proved that for any PITL formula an equivalent WS1S formula can be expressed.

We have implemented the tool PITL2MONA which translates PITL formulas to MONA specifications; this translation can be easily proved to be linear in the num-

2. We have not investigated other orderings

ber of symbols in the source formula. In this way a decision procedure for WS1S (MONA) can be applied to PITL. Furthermore, the decision procedure benefits from the optimisations included in MONA. To our knowledge, this is the first implementation of a decision procedure for PITL with *projection* and past operators that is based on automata. We have shown how PITL can be used, for example, in the specification of the dining-philosophers and a multimedia synchronisation problem. We have used PITL2MONA/MONA to verify a number of correctness properties over these specifications. A number of experimental results are shown which suggest the feasibility and convenience of automata-based decision procedures for PITL. Moreover, we have compared our approach with a tableau-based implementation of an equally expressive propositional subset; results show that MONA runs considerably faster, and that it can deal with more complex specifications. Interesting directions for further research include the identification of good variable orderings in the generation of MONA specifications, and a model-checking theory for PITL.

10. References

[BAS 00] BASIN D., FRIEDRICH S., "Combining WS1S and HOL", GABBAY D., DE RIJKE M., Eds., *Frontiers of Combining Systems 2*, vol. 7 of *Studies in Logic and Computation*, p. 39–56, Research Studies Press/Wiley, Baldock, Herts, UK, February 2000.

[BEE 01] BEER I., BEN-DAVID S., EISNER C., FISMAN D., GRINGAUZE A., RODEH Y., "The temporal logic Sugar", *CAV 2001*, LNCS, Springer-Verlag, 2001.

[BOW 98a] BOWMAN H., "An Interpretation of Cognitive Theory in Concurrency Theory", report num. 8-98, October 1998, Computing Laboratory, University of Kent at Canterbury.

[BOW 98b] BOWMAN H., THOMPSON S., "A Tableaux Method for Interval Temporal Logic with Projection", *International Conference on Analytic Tableaux and Related Methods (TABLEAUX'98)*, vol. 1397 of *Lecture Notes in AI*, Springer-Verlag, May 1998, p. 108-123.

[BOW 99] BOWMAN H., FACONTI G., "Analysing Cognitive Behaviour using LOTOS and MEXITL", *Formal Aspects of Computing*, vol. 11, 1999, p. 132-159, Springer.

[BOW 03a] BOWMAN H., CAMERON H., KING P., THOMPSON S., "Mexitl: Multimedia in Executable Interval Temporal Logic", *Formal Methods in System Design*, vol. 22, 2003, p. 5-38, Kluwer.

[BOW 03b] BOWMAN H., THOMPSON S., "A Decision Procedure and Complete Axiomatisation of Finite Interval Temporal Logic with Projection", *Journal of Logic and Computation*, vol. 13, num. 2, 2003, Oxford University Press.

[BRY 86] BRYANT R., "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computers*, vol. C-35, num. 8, 1986, p. 677-691.

[BÜC 60] BÜCHI J. R., "On a decision method in restricted second-order arithmetic", *Zeitschrift für Mathemathische Logik and Grundlagen der Mathematik*, vol. 6, 1960, p. 66-92.

[CHA 91] CHAOCHEN Z., HOARE C. A. R., RAVN A. P., "A Calculus of Durations", *Information Processing Letters*, vol. 40, num. 5, 1991, p. 269-276.

[DUA 96] DUAN Z. H., "An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming", PhD thesis, University of Newcastle Upon Tyne, May 1996.

[DUT 95] DUTERTRE B., "Complete Proof Systems for First Order Interval Temporal Logic", *Logic in Computer Science*, vol. 10, 1995, p. 36-43, IEEE.

[ELG 61] ELGOT C. C., "Decision Problems of Finite Automata Design and Related Arithmetics", *Trans. Amer. Math. Soc.*, vol. 98, 1961, p. 21-51.

[HAL 83] HALPERN J., MANNA Z., MOSZKOWSKI B., "A Hardware Semantics Based on Temporal Intervals", DIAZ J., Ed., *Proc. ICALP 1983*, vol. 154 of *Lecture Notes in Computer Science*, Springer-Verlag, 1983, p. 278-291.

[HAL 91] HALPERN J., SHOHAM Y., "A Propositional Modal Logic of Time Intervals", *Journal of the ACM*, vol. 38, num. 4, 1991, p. 935-962.

[HOL 01] HOLLANDER Y., MORLEY M., NOY A., "The *e* language: a fresh separation of concerns", *TOOLS Europe 2001*, Zurich, Switzerland, 2001, IEEE Computer Press.

[KLA 01] KLARLUND N., MÖLLER A., "MONA Version 1.4 User Manual", BRICS, University of Aarhus, Denmark, January 2001.

[KLA 02] KLARLUND N., MØLLER A., SCHWARTZBACH M. I., "MONA Implementation Secrets", *International Journal of Foundations of Computer Science*, vol. 13, num. 4, 2002, p. 571–586, World Scientific Publishing Company. Earlier version in Proc. 5th International Conference on Implementation and Application of Automata, CIAA '00, Springer-Verlag LNCS vol. 2088.

[KON 95] KONO S., "A Combination of Clausal and Non Clausal Temporal Logic Programs", FISHER M., OWENS R., Eds., *Executable Modal and Temporal Logics: Proc. of the IJCAI-93 Workshop*, p. 40-57, Springer, Berlin, Heidelberg, 1995.

[MAN 95] MANNA Z., PNUELI A., *Temporal verification of reactive systems: safety*, Springer, 1995.

[MEL 93] MELHAM T., "Higher Order Logic and Hardware Verification", *Cambridge Tracts in Theoretical Computer Science*, vol. 31, 1993, Cambridge University Press.

[MEY 75] MEYER A. R., "Weak monadic second-order theory of successor is not elementary recursive", PARIKH R., Ed., *Logic Colloquium (Proc. Symposium on Logic, Boston, 1972)*, vol. 453 of *Lecture Notes in Mathematics*, 1975, p. 132-154.

[MOS 83] MOSZKOWSKI B., "Reasoning about digital circuits", PhD thesis, Stanford University, 1983.

[MOS 85] MOSZKOWSKI B., "A Temporal Logic for Multilevel Reasoning about Hardware", *IEEE Computer*, vol. 18, num. 2, 1985, p. 10-19.

[MOS 86] MOSZKOWSKI B., *Executing Temporal Logic Programs*, Cambridge U. Press, 1986.

[MOS 00a] MOSZKOWSKI B., "An Automata-Theoretic Completeness Proof for Interval Temporal Logic", MONTANARI U., ROLIM J., WELZL E., Eds., *Proceedings of 27th International Colloquium on Automata, Languages and Programming (ICALP 2000)*, vol. 1853 of *Lecture Notes in Computer Science*, Geneva, Switzerland, July 9-15, 2000, Springer-Verlag, p. 223-234.

[MOS 00b] MOSZKOWSKI B., "A Complete Axiomatization of Interval Temporal Logic with Infinite Time", *Proceedings of Fifteenth Annual IEEE Symposium on Logic in Computer*

Science (LICS 2000), Santa Barbara, California, USA, June 26-29, 2000, IEEE Computer Society Press, p. 242-251.

[OWR 00] OWRE S., RUESS H., "Integrating WS1S with PVS", EMERSON E. A., SISTLA A. P., Eds., *Computer-Aided Verification, CAV '2000*, vol. 1855 of *Lecture Notes in Computer Science*, Chicago, IL, Jul. 2000, Springer-Verlag, p. 548–551.

[PAE 88] PAECH B., "Gentzen-Systems for Propositional Temporal Logics", BÖRGER E., BÜNING H. K., RICHTER M. M., Eds., *2nd Workshop on Computer Science Logic*, vol. 385 of *LNCS*, Duisburg, Germany, 1988, Springer.

[PAN 00a] PANDYA P. K., "DCVALID 1.4: The user manual", Tata Institute of Fundamental Research, Mumbay, India, September 2000, Available in `http://www.tcs.tifr.res.in/~pandya/dcvalid.html#paper`.

[PAN 00b] PANDYA P. K., "Specifying and deciding quantified discrete duration calculus formulae using DCVALID", report num. TCS00-PKP-1, 2000, Tata Institute of Fundamental Research.

[PNU 85] PNUELI A., "In transition from global to modular temporal reasoning about programs", *Volume F-13 of NATO Advanced Summmer Institutes*, Springer Verlag, 1985, p. 123-144.

[SAN 98] SANDHOLM A., SCHWARTZBACH M. I., "Distributed Safety Controllers for Web Services", ASTESIANO E., Ed., *Fundamental Approaches to Software Engineering, FASE'98, LNCS 1382*, num. 1382 Lecture Notes in Computer Science, Springer-Verlag, March/April 1998, p. 270-284, Also available as BRICS Technical Report RS-97-47.

[SMI 00] SMITH M. A., KLARLUND N., "Verification of a Sliding Window Protocol using IOA and MONA", BOLOGNESI T., LATELLA D., Eds., *Formal Methods for Distributed System Development*, p. 19-34, Kluwer Academic Publishers, 2000.

[THO 90] THOMAS W., "Automata on Infinite Objects", *Handbook of Theoretical Computer Science*, vol. B, p. 133-191, MIT Press, Elsevier, 1990.

[VAR 01] VARDI M., "Branching vs. linear time: the final showdown", *TACAS'01*, vol. 2031 of *LNCS*, Springer-Verlag, 2001.

[VEN 91] VENEMA Y., "A Modal Logic for Chopping Intervals", *Journal of Logic and Computation*, vol. 1, num. 4, 1991, p. 453-476.

A. PITL: the full set of operators

This section shows how PITL operators (future and past) are interpreted over intervals with past history, (σ, j), and their corresponding translation to $\mathcal{C}''$-structures. The satisfaction relation $\models$ over (σ, j) is defined as follows:

$(\sigma, j) \models v$ iff $v \in \sigma_j$

$(\sigma, j) \not\models$ **False**

$(\sigma, j) \models \neg P$ iff $(\sigma, j) \not\models P$

$(\sigma, j) \models P \vee Q$ iff $(\sigma, j) \models P \ \vee \ (\sigma, j) \models Q$

$(\sigma, j) \models \textbf{empty}$ iff $j = |\sigma|$

$(\sigma, j) \models \bigcirc P$ iff $j < |\sigma| \ \wedge \ (\sigma, j+1) \models P$

$(\sigma, j) \models P \textbf{ ; } Q$ iff $\exists k \in \mathbb{N}.\ j \leq k \leq |\sigma| \ \wedge \ ([\sigma]^k, j) \models P \ \wedge \ (\sigma, k) \models Q$

$(\sigma, j) \models P^*$ iff $j = 0 \ \vee \ \exists k_0, k_1, \ldots, k_m \in \mathbb{N}.$
$j = k_0 < k_1 < \ldots < k_m = |\sigma| \wedge$
$\forall i \in \mathbb{N}.\ 0 \leq i < m \Rightarrow ([\sigma]^{k_{i+1}}, k_i) \models P$

$(\sigma, j) \models P \textbf{ until } Q$ iff $\exists k \in \mathbb{N}.\ j \leq k \leq |\sigma| \wedge$
$(\sigma, k) \models Q \ \wedge \ \forall r \in \mathbb{N}.\ r < k \Rightarrow (\sigma, r) \models P$

$(\sigma, j) \models P \textbf{ proj } Q$ iff $j = 0 \ \vee \ \exists k_0, k_1, \ldots, k_m \in \mathbb{N}.$
$j = k_0 < k_1 < \ldots < k_m = |\sigma| \wedge$
$\forall i \in \mathbb{N}.\ 0 \leq i < m \Rightarrow ([\sigma]^{k_{i+1}}, k_i) \models P \wedge$
$(\sigma_0, \ldots, \sigma_{k_0}, \sigma_{k_1}, \ldots, \sigma_{k_m}, k_0) \models Q$

$(\sigma, j) \models P \ \widetilde{;} \ Q$ iff $\exists k \in \mathbb{N}.\ 0 \leq k \leq j \ \wedge \ (\sigma, k) \models P \wedge$
$((\sigma)^k, j-k) \models Q$

$(\sigma, j) \models P \, \mathcal{S} \, Q$ iff $\exists k \in \mathbb{N}.\ 0 \leq k \leq j \ \wedge \ (\sigma, k) \models Q \wedge$
$\forall r \in \mathbb{N}.\ k < r \leq j \Rightarrow (\sigma, r) \models P$

$(\sigma, j) \models \ominus P$ iff $j > 0 \ \wedge \ (\sigma, j-1) \models P$

The interpretation of PITL operators over $\mathcal{C}''$-structures is as follows:

$\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} v$ iff $(V, S) \in \mathcal{P} \ \wedge \ j \in S$

$\langle i, j, n, M, \mathcal{P} \rangle \not\models_{\mathcal{C}''} \textbf{False}$

$\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} \neg P$ iff $\langle i, j, n, M, \mathcal{P} \rangle \not\models_{\mathcal{C}''} P$

$\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} \bigcirc P$ iff $\exists k.\ cons(M, j, k) \wedge$
$\langle i, k, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} P$

$\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} P \vee Q$ iff $\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} P \vee$
$\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} Q$

$\langle i, j, n, M, \mathcal{P} \rangle \models_{\mathcal{C}''} \textbf{empty}$ iff $j = n$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \,;\, Q \quad \text{iff} \quad \exists k \in M.\, j \le k \le n \wedge \langle i,j,k,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q$$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P^* \quad \text{iff} \quad j = n \vee \exists k_0,k_1,\ldots,k_m \in M.\; j = k_0 < k_1 < \ldots < k_m = n \wedge \forall r.\, 0 \le r < m \Rightarrow \langle i,k_r,k_{r+1},M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \textbf{ until } Q \quad \text{iff} \quad \exists k \in M.\, j \le k \le n \wedge \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q \wedge \forall r \in M.\, j \le r < k \Rightarrow \langle i,r,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \textbf{ proj } Q \quad \text{iff} \quad j = n \vee \exists M' \subseteq M.\; M' = \{i,\ldots,k_0,k_1,\ldots,k_m\} \wedge \forall t.\, i \le t \le k_0 \Rightarrow (t \in M \Leftrightarrow t \in M') \wedge j = k_0 < k_1 < \ldots < k_m = n \wedge \forall r.\, 0 \le r < m \Rightarrow \langle i,k_r,k_{r+1},M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge \langle i,j,n,M',\mathcal{P}\rangle \models_{\mathcal{C}''} Q$$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \stackrel{\sim}{;} Q \quad \text{iff} \quad \exists k \in M.\, i \le k \le j \wedge \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge \langle k,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q$$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P\,\mathcal{S}\,Q \quad \text{iff} \quad \exists k \in M.\, i \le k \le j \wedge \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q \wedge \forall r \in M.\, k < r \le j \Rightarrow \langle i,r,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$$

$$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} \ominus P \quad \text{iff} \quad cons(M,k,j) \wedge \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$$

B. Theorems and proofs

THEOREM 10. — *For any interval σ and any PITL formula P the following holds:*

(Moszkowski's definition for $\sigma \models P^*$)
$\exists k_0,k_1,\ldots,k_m \in \mathbb{N}.\; k_0 = 0 \le k_1 \le \ldots \le k_m = |\sigma|$
$\wedge\, \forall i \in \mathbb{N}.\; 0 \le i < m \Rightarrow ([\sigma]^{k_{i+1}})^{k_i} \models P$

iff

(our definition for $\sigma \models P^*$)
$|\sigma| = 0 \vee$
$\exists k_0, k_1, \ldots, k_m \in \mathbb{N}.\ k_0 = 0 < k_1 < \ldots < k_m = |\sigma|$
$\wedge\ \forall i \in \mathbb{N}.\ 0 \leq i < m \Rightarrow ([\sigma]^{k_{i+1}})^{k_i} \models P$

PROOF. — (by induction on $|\sigma|$)

Case $|\sigma| = 0$:
In both definitions, P^* is trivially satisfiable by empty intervals.

Case $|\sigma| = n + 1,\ n \in \mathbb{N}$:
We will only prove the "sufficient" condition of the equivalence. The "necessary" condition trivially holds since a $<$-sequence is also a $\leq$-sequence. If σ is a non-empty interval and indeed there exists such a $\leq$-sequence, then there must be an iteration of P holding over a non-empty suffix of the model. In other words, then, there must exist $j,\ 1 \leq j \leq m$ such that $k_{j-1} < k_j = \ldots = k_m = |\sigma|$ and $(\sigma)^{k_{j-1}} \models P$. Otherwise there is no way to link the first and last state of σ with a joint sequence of models for P. Furthermore,

$\exists k_0, k_1, \ldots, k_j \in \mathbb{N}.\ k_0 = 0 \leq k_1 \leq \ldots \leq k_{j-1} < k_j = |\sigma|$
$\wedge\ \forall i \in \mathbb{N}.\ 0 \leq i < j \Rightarrow ([\sigma]^{k_{i+1}})^{k_i} \models P$

As $k_{j-1} \leq n$ we can use the induction hypothesis on the the $\leq$-prefix of the previous sequence (i.e. the points $k_0, \ldots, k_{j-1}$), to obtain the required expression:

$\exists k_0, k_1, \ldots, k_j \in \mathbb{N}.\ k_0 = 0 < k_1 < \ldots < k_{j-1} < k_j = |\sigma|$
$\wedge\ \forall i \in \mathbb{N}.\ 0 \leq i < j \Rightarrow ([\sigma]^{k_{i+1}})^{k_i} \models P$

■

LEMMA 11. — *Let $\sigma \in \mathcal{I}$ and $\Theta(\sigma) = \langle 0, |\sigma|, \mathcal{P} \rangle$. Then, for any PITL^0 formula F the following holds:*

$$\langle i, n, \mathcal{P} \rangle \models_{\mathcal{C}} F \textit{ iff } ([\sigma]^n)^i \models F$$

PROOF. — The proof is by induction on $|F|$.

Case $|F| = 1$:

$\langle i, n, \mathcal{P} \rangle \models_{\mathcal{C}} v$	iff	[def. of $\models_{\mathcal{C}}$]
$(V, S) \in \mathcal{P} \ \wedge\ i \in S$	iff	[def. of $\mathcal{P}$]
$v \in \sigma_i$	iff	[def. of $\models$]
$([\sigma]^n)^i \models v$		

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}}$ **empty** iff [def. of $\models_{\mathcal{C}}$]
$i = n$ iff [def. of $[]^i$, $()^i$]
$|([\sigma]^n)^i| = 0$ iff [def. of $\models$]
$([\sigma]^n)^i \models$ **empty**

By definition, $\langle i, n, \mathcal{P}\rangle \not\models_{\mathcal{C}}$ **False** and $([\sigma]^n)^i \not\models$ **False**

Case $|F| > 1$:

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} \neg P$ iff [def. of $\models_{\mathcal{C}}$]
$\langle i, n, \mathcal{P} \not\models_{\mathcal{C}} P$ iff [hyp. on $|P| < |F|$]
$([\sigma]^n)^i \not\models P$ iff [def. of $\models$]
$([\sigma]^n)^i \models \neg P$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} \bigcirc P$ iff [def. of $\models_{\mathcal{C}}$]
$i < n \ \wedge \ \langle i+1, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$ iff [hyp. on $|P| < |F|$]
$([\sigma]^n)^{i+1} \models P$ iff [def. of $\models$]
$([\sigma]^n)^i \models \bigcirc P$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P \vee Q$ iff [def. of $\models_{\mathcal{C}}$]
$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P \vee \langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$]
$([\sigma]^n)^i \models P \ \vee \ ([\sigma]^n)^i \models Q$ iff [def. of $\models$]
$([\sigma]^n)^i \models P \vee Q$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P \,;\, Q$ iff [def. of $\models_{\mathcal{C}}$]
$\exists k.\, \langle i, k, \mathcal{P}\rangle \models_{\mathcal{C}} P \ \wedge \ \langle k, n, \mathcal{P}\rangle \models_{\mathcal{C}} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$]
$\exists k.\, ([\sigma]^k)^i \models P \ \wedge \ ([\sigma]^n)^k \models Q$ iff [def. of $\models$]
$([\sigma]^n)^i \models P \,;\, Q$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P^*$ iff [def. of $\models_{\mathcal{C}}$]
$i = n \vee$
$\exists k_0, k_1, \ldots, k_m.\, i = k_0 < k_1 < \ldots < k_m = n$
$\wedge \forall j.\, 0 \leq j < m \Rightarrow \langle k_j, k_{j+1}, \mathcal{P}\rangle \models_{\mathcal{C}} P$ iff [hyp. on $|P| < |F|$]
$i = n \vee$
$\exists k_0, k_1, \ldots, k_m.\, i = k_0 < k_1 < \ldots < k_m = n$
$\wedge \forall j.\, 0 \leq j < m \Rightarrow ([\sigma]^{k_{j+1}})^{k_j} \models P$ iff [def. of $\models$]
$([\sigma]^n)^i \models P^*$

$\langle i, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$ **until** Q iff [def. of $\models_{\mathcal{C}}$]
$\exists k.\, i \leq k \leq n \ \wedge\ \langle k, n, \mathcal{P}\rangle \models_{\mathcal{C}} Q \ \wedge$
$\forall j.\, i \leq j < n \Rightarrow \langle j, n, \mathcal{P}\rangle \models_{\mathcal{C}} P$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$]
$\exists k.\, i \leq k \leq n \ \wedge\ ([\sigma]^n)^k \models Q \ \wedge$
$\forall j.\, i \leq j < n \Rightarrow ([\sigma]^n)^j \models P$ iff [def. of $\models$]
$([\sigma]^n)^i \models P$ **until** Q

■

THEOREM 12. — *For any $\sigma \in \mathcal{I}$ and any PITL^0 formula F, $\Theta(\sigma) \models_{\mathcal{C}} F$ iff $\sigma \models F$*

PROOF. — Theorem 12 is a corollary of Lemma 11. ■

LEMMA 13. — *Let $\sigma \in \mathcal{I}$ and $\Theta'(\sigma) = \langle 0, |\sigma|, \{0, \ldots, |\sigma|\}, \mathcal{P}\rangle$. Let $M \subseteq \{0, \ldots, |\sigma|\}$ and $i, n, t_0, t_1, \ldots, t_s \in M$ s.t. $i = t_0 < t_1 < \ldots < t_s = n$ and $\forall r \in \mathbb{N}.\, 0 \leq r < s \Rightarrow \neg\exists k \in M.\, t_r < k < t_{r+1}$. Then, for any PITL^1 formula F the following holds:*

$$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} F \text{ iff } \sigma_{t_0}, \ldots, \sigma_{t_s} \models F$$

PROOF. — The proof is by induction on $|F|$

Case $|F| = 1$:

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} v$ iff [def. of $\models_{\mathcal{C}'}$]
$(V, S) \in \mathcal{P} \ \wedge\ i \in S$ iff [def. of $\mathcal{P}$]
$v \in \sigma_i$ iff [$i = t_0$]
$v \in \sigma_{t_0}$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models v$

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'}$ **empty** iff [def. of $\models_{\mathcal{C}'}$]
$i = n$ iff [$i = t_0,\ n = t_s$]
$|\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle| = 0$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models$ **empty**

By definition $\langle i, n, M, \mathcal{P}\rangle \not\models_{\mathcal{C}'}$ **False** and $\sigma_{t_0}, \ldots, \sigma_{t_s} \not\models$ **False**

Case $|F| > 1$:

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} \neg P$ iff [def. of $\models_{\mathcal{C}'}$]
$\langle i, n, M, \mathcal{P}\rangle \not\models_{\mathcal{C}'} P$ iff [hyp. on $|P| < |F|$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \not\models P$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models \neg P$

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} \bigcirc P$ iff [def. of $\models_{\mathcal{C}'}$]
$\exists j \in M.\ cons(M, i, j) \wedge \langle j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P$ iff [hyp. on $|P| < |F|$, $i = t_0$, $cons(M, t_0, t_1)$, def. of M]
$\langle \sigma_{t_1}, \ldots, \sigma_{t_s}\rangle \models P$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models \bigcirc P$

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P \vee Q$ iff [def. of $\models_{\mathcal{C}'}$]
$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P \ \vee\ \langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models P \ \vee\ \langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models Q$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models P \vee Q$

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P\,;Q$ iff [def. of $\models_{\mathcal{C}'}$]
$\exists k \in M.\ i \le k \le n \wedge$
$\langle i, k, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P \wedge \langle k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$, $i = t_0,\ n = t_s$, def. of M]
$\exists k \in M.\ t_0 \le k \le t_s \wedge$
$\langle \sigma_{t_0}, \ldots, \sigma_k\rangle \models P \wedge \langle \sigma_k, \ldots, \sigma_{t_s}\rangle \models Q$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models P\,;Q$

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P^*$ iff [def. of $\models_{\mathcal{C}'}$]
$i = n \vee$
$\exists k_0, k_1, \ldots, k_m \in M.\ i = k_0 < k_1 < \ldots < k_m = n \wedge$
$\forall j.\ 0 \le j < m \Rightarrow \langle k_j, k_{j+1}, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P$ iff [hyp. on $|P| < |F|$, $i = t_0,\ n = t_s$, def. of M]
$t_0 = t_s \vee$
$\exists k_0, k_1, \ldots, k_m \in M.\ t_0 = k_0 < k_1 < \ldots < k_m = t_s \wedge$
$\forall j.\ 0 \le j < m \Rightarrow \langle \sigma_{k_j}, \ldots, \sigma_{k_{j+1}}\rangle \models P$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models P^*$

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P$ **until** Q iff [def. of $\models_{\mathcal{C}'}$]
$\exists k \in M.\ i \le k \le n\ \wedge\ \langle k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} Q\ \wedge$
$\forall j \in M.\ i \le j < k \Rightarrow \langle j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$, $i = t_0,\ n = t_s$, def. of M]
$\exists k \in M.\ t_0 \le k \le t_s \wedge\ \langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models_{\mathcal{C}'} Q\ \wedge$
$\forall j \in M.\ t_0 \le j < k \Rightarrow \langle \sigma_j, \ldots, \sigma_{t_s}\rangle \models_{\mathcal{C}'} P$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models P$ **until** Q

$\langle i, n, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P$ **proj** Q iff [def. of $\models_{\mathcal{C}'}$]
$i = n\ \vee\ \exists M' = \{k_0, k_1, \ldots, k_m\} \subseteq M.$
$i = k_0 < k_1 < \ldots < k_m = n\ \wedge$
$\forall j.\ 0 \le j < m \Rightarrow \langle k_j, k_{j+1}, M, \mathcal{P}\rangle \models_{\mathcal{C}'} P\ \wedge$
$\langle i, n, M', \mathcal{P}\rangle \models_{\mathcal{C}'} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$, $i = t_0,\ n = t_s$, def. of M]
$t_0 = t_s\ \vee\ \exists M' = \{k_0, k_1, \ldots, k_m\} \subseteq M.$
$t_0 = k_0 < k_1 < \ldots < k_m = t_s\ \wedge$
$\forall j.\ 0 \le j < m \Rightarrow \langle \sigma_{k_j}, \ldots, \sigma_{k_{j+1}}\rangle \models P\ \wedge$
$\langle \sigma_{k_0}, \ldots, \sigma_{k_m}\rangle \models Q$ iff [def. of $\models$]
$\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle \models P$ **proj** Q

■

THEOREM 14. — *For any $\sigma \in \mathcal{I}$ and for any PITL[1] formula F,*

$$\Theta'(\sigma) \models_{\mathcal{C}'} F \text{ iff } \sigma \models F$$

PROOF. — Theorem 14 is a corollary of Lemma 13. ■

LEMMA 15. — *Let $\sigma \in \mathcal{I}$ and $\Theta''(\sigma) = \langle 0, 0, |\sigma|, \{0, \ldots, |\sigma|\}, \mathcal{P}\rangle$. Let $M \subseteq \{0, \ldots, |\sigma|\}$ and $i, j, n, t_0, t_1, \ldots, t_s \in M$ s.t. $i = t_0 < t_1 < \ldots < t_s = n$, $j = t_u,\ 0 \le u \le s$ and $\forall r \in \mathbb{N}.\ 0 \le r < s \Rightarrow \neg\exists k \in M.\ t_r < k < t_{r+1}$. Then, for any PITL formula F the following holds:*

$$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} F \text{ iff } (\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models F$$

PROOF. — The proof is by induction on $|F|$.

Case $|F| = 1$:

$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} v$ iff [def. of $\models_{\mathcal{C}''}$]
$(V, S) \in \mathcal{P} \ \wedge \ j \in S$ iff [def. of $\mathcal{P}$]
$v \in \sigma_j$ iff [$j = t_u$]
$v \in \sigma_{t_u}$ iff [def. of $\models$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models v$

$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''}$ **empty** iff [def. of $\models_{\mathcal{C}''}$]
$j = n$ iff [$j = t_u,\ n = t_s$]
$|\langle \sigma_{t_u}, \ldots, \sigma_{t_s}\rangle| = 0$ iff [def. of $\models$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models$ **empty**

By definition $\langle i, j, n, M, \mathcal{P}\rangle \not\models_{\mathcal{C}''}$ **False** and $(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \not\models$ **False**

Case $|F| > 1$:

$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} \neg P$ iff [def. of $\models_{\mathcal{C}''}$]
$\langle i, j, n, M, \mathcal{P}\rangle \not\models_{\mathcal{C}''} P$ iff [hyp. on $|P| < |F|$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \not\models P$ iff [def. of $\models$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models \neg P$

$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} \bigcirc P$ iff [def. of $\models_{\mathcal{C}''}$]
$\exists k \in M.\ cons(M, j, k) \ \wedge \ \langle i, k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P$ iff [hyp. on $|P| < |F|$, $j = t_u$, $cons(M, t_u, t_{u+1})$, def. of M]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_{u+1}}, \ldots, \sigma_{t_s}\rangle, t_{u+1} - t_0) \models_{\mathcal{C}''} P$ iff [def. of $\models$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models \bigcirc P$

$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P \vee Q$ iff [def. of $\models_{\mathcal{C}''}$]
$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P \vee$
$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models P \vee$
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models Q$ iff [def. of $\models$]
$(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models P \vee Q$

$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \,;\, Q$ iff [def. of $\models_{\mathcal{C}''}$]
$\exists k \in M.\ j \le k \le n \wedge$
$\langle i,j,k,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge$
$\langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q$ iff [hyp. on $|P|<|F|$, $|Q|<|F|$, $j=t_u,\ n=t_s$, def. of M]

$\exists k \in M.\ t_u \le k \le t_s \wedge$
$(\langle \sigma_{t_0},\ldots,\sigma_{t_u},\ldots,\sigma_k\rangle, t_u - t_0) \models P \wedge$
$(\langle \sigma_{t_0},\ldots,\sigma_k,\ldots,\sigma_{t_s}\rangle, k - t_0) \models Q$ iff [def. of $\models$]
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models P \,;\, Q$

$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P^*$ iff [def. of $\models_{\mathcal{C}''}$]
$j = n \vee$
$\exists k_0,k_1,\ldots,k_m \in M.\ j = k_0 < k_1 < \ldots < k_m = n \wedge$
$\forall r.\ 0 \le r < m \Rightarrow \langle i,k_r,k_{r+1},M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$ iff [hyp. on $|P|<|F|$, $j=t_u$, $n=t_s$, def. of M]

$t_u = t_s \vee$
$\exists k_0,k_1,\ldots,k_m \in M.\ t_u = k_0 < k_1 < \ldots < k_m = t_s \wedge$
$\forall r.\ 0 \le r < m \Rightarrow (\langle \sigma_{t_0},\ldots,\sigma_{k_r},\ldots,\sigma_{k_{r+1}}\rangle, k_r - t_0) \models P$ iff [def. of $\models$]
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models P^*$

$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$ **until** Q iff [def. of $\models_{\mathcal{C}''}$]
$\exists k \in M.\ j \le k \le n \ \wedge\ \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q \wedge$
$\forall r \in M.\ j \le r < k \Rightarrow \langle i,r,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$ iff [hyp. on $|P|<|F|$, $|Q|<|F|$, $j=t_u,\ n=t_s$, def. of M]

$\exists k \in M.\ t_u \le k \le t_s \wedge$
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models_{\mathcal{C}''} Q \wedge$
$\forall r \in M.\ t_u \le r < k \Rightarrow$
$(\langle \sigma_{t_0},\ldots,\sigma_r,\ldots,\sigma_{t_s}\rangle, r - t_0) \models_{\mathcal{C}''} P$ iff [def. of $\models$]
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models P$ **until** Q

$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \textbf{ proj } Q$ iff [def. of $\models_{\mathcal{C}''}$]
$j = n \ \vee \ \exists M' \subseteq M.$
$M' = \{i,\ldots,k_0,k_1,\ldots,k_m\} \wedge$
$\forall t.\, i \leq t \leq k_0 \Rightarrow (t \in M \Leftrightarrow t \in M') \wedge$
$j = k_0 < k_1 < \ldots < k_m = n \wedge$
$\forall r.\, 0 \leq r < m \Rightarrow \langle k_r, k_{r+1}, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P \wedge$
$\langle i,j,n,M',\mathcal{P}\rangle \models_{\mathcal{C}''} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$, $j = t_u,\ n = t_s$, def. of M]

$t_u = t_s \ \vee \ \exists M' \subseteq M.$
$M' = \{i,\ldots,k_0,k_1,\ldots,k_m\} \wedge$
$\forall t.\, i \leq t \leq k_0 \Rightarrow (t \in M \Leftrightarrow t \in M') \wedge$
$t_u = k_0 < k_1 < \ldots < k_m = t_s \wedge$
$\forall r. 0 \leq r < m \Rightarrow (\langle \sigma_{t_0},\ldots,\sigma_{k_r},\sigma_{k_{r+1}}\rangle, k_r - t_0) \models P \wedge$
$(\langle \sigma_{t_0},\ldots,\sigma_{k_0},\ldots,\sigma_{k_m}\rangle, t_u - t_0) \models Q$ iff [def. of $\models$]
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models P \textbf{ proj } Q$

$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \ \widetilde{;} \ Q$ iff [def. of $\models_{\mathcal{C}''}$]
$\exists k \in M.\, i \leq k \leq j \wedge$
$\langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \ \wedge \ \langle k,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$, $j = t_u,\ n = t_s$, def. of M]

$\exists k \in M.\, i \leq k \leq t_0 \wedge$
$(\langle \sigma_{t_0},\ldots,\sigma_k,\ldots,\sigma_{t_s}\rangle, k - t_0) \models P \wedge$
$(\sigma_k,\ldots,\sigma_{t_s}\rangle, t_u - k) \models Q$ iff [def. of $\models$]
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models P \ \widetilde{;} \ Q$

$\langle i,j,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P \,\mathcal{S}\, Q$ iff [def. of $\models_{\mathcal{C}''}$]
$\exists k \in M.\, i \leq k \leq j \ \wedge \ \langle i,k,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} Q \wedge$
$\forall\, r \in M.\, k < r \leq j \Rightarrow \langle i,r,n,M,\mathcal{P}\rangle \models_{\mathcal{C}''} P$ iff [hyp. on $|P| < |F|$, $|Q| < |F|$, $j = t_u,\ n = t_s$, [def. of M]

$\exists k \in M.\, i \leq k \leq t_u \wedge$
$(\langle \sigma_{t_0},\ldots,\sigma_k,\ldots,\sigma_{t_s}\rangle, k - t_0) \models Q \wedge$
$\forall r \in M.\, k < r \leq t_u \Rightarrow$
$(\langle \sigma_{t_0},\ldots,\sigma_r,\ldots,\sigma_{t_s}\rangle, r - t_0) \models P$ iff [def. of $\models$]
$(\langle \sigma_{t_0},\ldots,\sigma_{t_s}\rangle, t_u - t_0) \models P \,\mathcal{S}\, Q$

$\langle i, j, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} \ominus P$	iff	[def. of $\models_{\mathcal{C}''}$]
$cons(M, k, j) \wedge \langle i, k, n, M, \mathcal{P}\rangle \models_{\mathcal{C}''} P$	iff	[hyp. on $\lvert P\rvert < \lvert F\rvert$, $\lvert Q\rvert < \lvert F\rvert$, $j = t_u,\ n = t_s$, [def. of M]
$cons(M, t_{u-1}, t_u) \wedge$ $(\langle \sigma_{t_0}, \ldots, \sigma_{t_{u-1}}, \sigma_{t_u}, \ldots, \sigma_{t_s}\rangle, t_{u-1} - t_0) \models P$ $(\langle \sigma_{t_0}, \ldots, \sigma_{t_s}\rangle, t_u - t_0) \models \ominus P$	iff	[def. of $\models$]

■

THEOREM 16. — *For any $\sigma \in \mathcal{I}$ and for any PITL formula F,*

$$\Theta''(\sigma) \models_{\mathcal{C}''} F \textit{ iff } (\sigma, 0) \models F$$

PROOF. — Theorem 16 is a corollary of Lemma 15. ■