Temporal Logic as a Programming Language

Dissertation an der Fakultät für Mathematik
der Ludwig-Maximilians-Universität München
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

von

Stephan Merz

Berichterstatter: Prof. Dr. Fred Kröger und
Prof. Dr. Martin Wirsing

eingereicht am: 09. Juni 1992
Tag der mündlichen Prüfung: 22. Juli 1992

Abstract

The central topic of this thesis is the characterization of a class of temporal logic formulas for which a model can be constructed automatically and efficiently. This study is motivated by the desire to close the gap between the "declarative" use of temporal logic as a specification formalism, especially for reactive systems, and the implementation of such systems in a different "operational" formalism. The logical framework employed is a temporal logic based on fixpoint operators, allowing the results to be as general as practical by treating various temporal operators in a uniform way. A characterization of safety properties in this framework is given, which is then specialized to a similar characterization of deterministic safety properties. Starting from such formulas, the theory of "executable" temporal logic formulas is at first developed for propositional temporal logic, aiming at the characterization of formulas which allow for a model construction using a backtracking-free and future-directed algorithm whose time complexity is linear in the length of the input formulas. For the first-order case, first some new incompleteness results are presented which further motivate the preference of a model-construction over a "proofs-as-programs" paradigm for the design of a programming language based on temporal logic. The framework developed for propositional temporal logic is extended to first-order temporal logic. Finally, the parallel composition of modules is studied using a conventional interleaving semantics and, in particular, the transition from properties of single modules to properties of composed systems is addressed. For programs expressed in the formalism developed before, a rule is given which allows the derivation of unconditional system properties in the presence of mutual dependencies between the modules.

Acknowledgement

I am very grateful to Prof. Kröger for suggesting to me the topic of this study, for creating a very pleasant working atmosphere, and for his guidance along the way. I am also indebted to Prof. Wirsing for his prompt willingness to act as the second referee. Special thanks are also due to Hans Gaßner, Frank Leßke and Holger Schlingloff for careful proofreading and suggesting several improvements in the exposition.

Financially, my work has been supported by a scholarship of the Studienstiftung des deutschen Volkes whose generosity is gratefully acknowledged. In particular, I wish to express my thanks to my advisor there, Prof. Scholz.

Contents

List of Figures

Chapter 1

Introduction

The development of formal methods supporting the specification and design of reactive systems has been among the most active research areas of computer science in recent years. In contrast to more conventional transformational programs whose purpose is to produce some result from initially given input data, reactive systems continually have to react to requests presented to them by their environment. At least ideally, reactive systems are not supposed to terminate at any time. Examples for reactive systems include operating systems, database servers, process control applications, reservation systems and so forth. Closely connected with the notion of reactivity is the notion of concurrency: At the very least, the system and its environment proceed concurrently, maintaining a continuing interaction. Frequently, it may be found convenient or necessary to implement the reactive system by a collection of independent modules which may even operate on separate processing entities, increasing the degree of concurrency of the system. It is well known by theoreticians and practitioners alike that correct concurrent programs are much harder to develop and possess types of errors, such as deadlock or starvation, which are unknown in sequential programs but arise from the independent execution of concurrent modules. Therefore, formal methods for the specification, development and verification of reactive systems are probably still more important than for sequential, transformational programs.

Obviously, reactive systems cannot be modelled as computing a relation between input and output data. Rather, potentially infinite sequences of states assumed or of actions performed by the system and its environment have to be considered for the description of the system's behaviour. Several formalisms have been proposed to aid in the design of reactive systems. One of the most successful tools for reasoning about behaviours of reactive systems is *temporal logic* introduced in Kröger [29] or Manna & Pnueli [38]. It assumes runs of reactive systems to be represented by infinite sequences of states and combines classical (propositional or first-order) logic for assertions concerning single states with new temporal operators expressing assertions relating several states. Temporal logic can be used to state properties desired of a reactive system, i.e., for specification. It can also be used for verifying the correctness of reactive systems, assuming that the runs of programs expressed in a given programming language can be characterized by temporal logic axioms. Properties of the particular reactive system are deduced from these axioms using temporal reasoning, establishing that every run of the program is a model of the given specification.

Typical development methods such as the UNITY formalism presented in Chandy & Misra [14] proceed by "refining" the initial specification, i.e., producing more and more detailed intermediate specifications (where at every step it has to be ensured that the new specification implies the original one, i.e., that every implementation of the new specification also implements the original specification) until an implementation can be directly "read off" the final specification. The implementation is then verified w.r.t. this final specification. However, it is usually not very clear what it means for a specification to be more detailed than another one. Similarly, the transition from the final specification to the implementation is often informal (else, the final verification step might be unnecessary). The central issue addressed in this thesis will be the identification of a class of temporal logic formulas which can be executed directly, thus eliminating the need for a separate implementation language as well as the need for the final verification step. The main benefit of such an approach is to free the developer from having to cope with different languages for specification, proofs and implementation, but to offer one language, viz. temporal logic, covering all three areas. In particular, a program implementing a reactive system is simply a set of temporal logic formulas. One of the main problems of logic-based programming paradigms in general, which will also have to be addressed here, is to ensure the consistency of the final specification — since inconsistent sets of formulas imply every assertion (in particular, every property stated in the specification of the reactive system to develop), but they have no models, i.e., attempting their execution will fail. The idea of directly executing temporal logic formulas has also been pursued by other authors, most prominently by Moszkowski [41] and by Barringer et al. [11] in the MetateM project. Our approach differs by offering high-level programming constructs while ensuring the effective and forward-directed execution of temporal logic programs. A more detailed discussion of the similarities and differences with respect to other formalisms will be found in the conclusion.

Modularity is another requirement for development formalisms. In particular, large systems will be broken up into smaller modules which proceed largely independently, interacting and maybe synchronizing to produce the desired behaviour of the system. Then, properties of the entire system have to be derived from properties of the single modules. This becomes particularly delicate if the correctness of every single module depends on the other modules meeting certain assumptions.

Throughout the following, attention will be restricted to asynchronous systems without any real-time requirements. Although this may seem unrealistic and overly restrictive, most problems concerning the correctness of concurrent programs can be studied in this framework. Temporal logics for real-time systems have been introduced, e.g., in Alur & Henzinger [9].

Concurrency will always be modelled by interleaving runs of single modules. This approach appears, on one hand, somewhat simplistic since it shields every module from interference by simultaneous actions of different modules of the system. However, it has frequently been pointed out that such difficulties can be overcome by choosing a suitable grain of atomicity. On the other hand, it introduces additional difficulties by allowing unnatural runs of the composed system where, e.g., some module never makes a step even if it could do so. If such runs are undesired, they may be ruled out by stating additional fairness requirements for the progress of the individual modules. Recently, temporal logics have been proposed which are based on the so-called "real-concurrency" paradigm, e.g., Katz & Peled [24] or Paech [42]. Such logics may offer a higher degree of compositionality, i.e., a smooth transition from module properties to system properties. However, the price to be paid is usually a weaker notion of correctness and a less intuitive model of runs of reactive systems.

The outline of this thesis is as follows: Chapter 2 introduces basic notions concerning properties of reactive systems as well as the temporal logic νTL which will be used subsequently. Chapter 3 studies some questions concerning the axiomatization of the logic νTL. Chapter 4 deals with the execution of propositional νTL programs, in particular defining a class of executable temporal logic formulas and giving an "interpreter algorithm" constructing models of sets of such formulas. The extension of this framework to the first-order case is studied in chapter 5. The subject of chapter 6 are modular systems composed from individual modules. Chapter 7 gives two examples applying the developed theory to concrete specifications, and chapter 8 concludes with discussing the results and hinting at future work necessary for obtaining a full-fledged framework for developing reactive systems.

Notational conventions

In the sequel, $\mathbb{N}$ denotes the set $\{0, 1, 2, \ldots\}$ of natural numbers, and ω denotes the smallest infinite ordinal number where $n < \omega$ holds for every $n \in \mathbb{N}$. Let Σ be a non-empty set. Σ^* and Σ^ω denote the sets of finite and infinite sequences, respectively, of elements of Σ (Σ-sequences, for short), formally defined as mappings from an initial segment of $\mathbb{N}$ to σ. In particular, ε denotes the empty sequence. $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ denotes the set of non-empty finite Σ-sequences. Let $\sigma = (s_0, s_1, \ldots) \in \Sigma^* \cup \Sigma^\omega$ be a Σ-sequence and $i \in \mathbb{N}$, then

- the length $|\sigma|$ of σ is defined as $|\sigma| = 0$ if $\sigma = \varepsilon$ is the empty sequence, $|\sigma| = n + 1$ if $\sigma = (s_0, \ldots, s_n) \in \Sigma^+$ is a finite non-empty sequence, and $|\sigma| = \omega$ if $\sigma \in \Sigma^\omega$ is infinite,
- for $0 \leq i < |\sigma|$, $\sigma[i]$ denotes the i-th element s_i of σ,
- for $0 \leq i < |\sigma|$, $\sigma[..i]$ denotes the finite sequence $(s_0, s_1, \ldots, s_i) \in \Sigma^+$ (the *prefix* of σ up to and including s_i); $\sigma[..\text{-}1]$ is taken to denote the empty sequence ε,
- for $0 \leq i < |\sigma|$, $\sigma[i..]$ denotes the sequence $(s_i, s_{i+1}, \ldots) \in \Sigma^* \cup \Sigma^\omega$ (the *suffix* of σ starting at state s_i).

Let $\sigma \in \Sigma^*$ and $\tau \in \Sigma^* \cup \Sigma^\omega$ be two Σ-sequences. σ is called a *prefix* of τ (denoted by $\sigma \preceq \tau$) if $|\tau| \geq |\sigma|$ and $\sigma[i] = \tau[i]$ for $0 \leq i < |\sigma|$. In particular, $\varepsilon \preceq \tau$ holds for every Σ-sequence τ. If $\sigma \preceq \tau$ and $|\sigma| < |\tau|$, σ is a *strict prefix* of τ (denoted by $\sigma \prec \tau$). For infinite sequences $\sigma, \tau \in \Sigma^\omega$, $\sigma \preceq \tau$ holds iff $\sigma = \tau$. For $\sigma \in \Sigma^*$ and $\tau \in \Sigma^* \cup \Sigma^\omega$, the *product* $\sigma\tau \in \Sigma^* \cup \Sigma^\omega$ is defined by $|\sigma\tau| = \omega$ if $|\tau| = \omega$, $|\sigma\tau| = |\sigma| + |\tau|$ otherwise, $(\sigma\tau)[i] = \sigma[i]$ for $0 \leq i < |\sigma|$ and $(\sigma\tau)[i] = \tau[i - |\sigma|]$ for $|\sigma| \leq i \leq |\sigma\tau|$. Similarly, for $s \in \Sigma$ and $\sigma \in \Sigma^*$, $\sigma s \in \Sigma^+$ denotes the sequence where $|\sigma s| = |\sigma| + 1$, $(\sigma s)[i] = \sigma[i]$ for $0 \leq i < |\sigma|$, and $(\sigma s)[|\sigma|] = s$.

The powerset of a set M, i.e., the set of subsets of M, is denoted by 2^M.

Let $R \subseteq M \times N$ be a binary relation (where M and N are sets). For any $x \in M$, $R(x) = \{y \mid (x, y) \in R\}$ denotes the image of x under R. If $M = N$, the relations R^* and R^+ denote the reflexive transitive closure and the transitive closure of R, respectively.

Chapter 2

Properties and temporal logic

2.1 Properties of reactive systems

This section presents some general results about properties of reactive systems. Most of this material can also be found, e.g., in Manna & Pnueli [37], or Alpern & Schneider [7]. However, the proofs given here are intended to be as elementary as possible.

Let Σ be a set (of *states*). Informally, Σ may be thought of as the set of states a reactive system may assume. A run of the system can be represented as an infinite sequence $\sigma = (s_0, s_1, \dots) \in \Sigma^\omega$. A ($\Sigma-$)*property* P is any set $P \subseteq \Sigma^\omega$ of runs. To introduce some structure into the class of properties, two basic classes of properties are usually distinguished in the literature[1], viz. *safety properties* and *liveness properties* which are formally defined as follows:

Definition 2.1 (safety and liveness properties)

- *A property $P \subseteq \Sigma^\omega$ is a* safety property *iff for every $\sigma \in \Sigma^\omega$,*

$$\sigma \in P \quad \Longleftrightarrow \quad \forall i \geq 0 \exists \tau \in \Sigma^\omega (\sigma[..i]\tau \in P).$$

- *A property $P \subseteq \Sigma^\omega$ is a* liveness property *iff*

$$\forall \sigma \in \Sigma^* \exists \tau \in \Sigma^\omega (\sigma\tau \in P).$$

Notice that in the definition of safety properties, only the "$\Longleftarrow$" part is non-trivial.

[1] Manna & Pnueli [37], however, introduce a hierarchy of properties which agrees with the classification given here on the safety properties.

The intuition behind these definitions is given by the following informal characterizations: P is a safety property if for every run σ *not* contained in P there is a finite prefix $\sigma[..i]$ which can not be completed by any sequence $\tau \in \Sigma^\omega$ to obtain a run $\sigma[..i]\tau$ in P. Stated differently, for every $\sigma \notin P$ something "bad" must have happened after some finite number of steps which cannot be remedied by any future (well-)behavior. Hence, in LAMPORT'S [31] popular characterization, safety properties express that "something bad never happens". A liveness property P, on the other hand, can never be refuted by observing only a finite prefix of some run. In contrast, every finite state sequence can be completed to yield a run in P, hence P states that, again in the words of LAMPORT, "something good eventually happens". Notice, however, that in contrast to the "bad thing" referred to above, the occurrence of the "good thing" does not have to be observable in finite time.

A simple example of a safety property P is given by the set of runs such that every occurrence of some designated state $s \in \Sigma$ is *immediately* followed by the occurrence of some other designated state t, i.e.,

$$P = \{\sigma \in \Sigma^\omega \mid \forall i : \sigma[i] = s \;\Rightarrow\; \sigma[i+1] = t\}.$$

A similarly defined liveness property Q is the set of runs such that every occurrence of s is *eventually* followed by t, i.e.,

$$Q = \{\sigma \in \Sigma^\omega \mid \forall i : \sigma[i] = s \;\Rightarrow\; \exists j > i \; (\sigma[j] = t)\}.$$

Surprisingly, these two simple notions already yield some fundamental facts about properties of reactive systems which will be the subject of this section.

Lemma 2.2

(i) *Let $\{P_i \mid i \in I\}$ be a set of safety properties. Then $\bigcap_{i \in I} P_i$ is a safety property.*

(ii) *Let $P_1, \ldots, P_n$ be safety properties. Then $\bigcup_{i=1}^{n} P_i$ is a safety property.*

Proof:

(i) If $I = \emptyset$, $\bigcap_{i \in I} P_i = \Sigma^\omega$ which clearly is a safety property. Else, assume $\sigma \in \Sigma^\omega$ to be a state sequence such that for every $j \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $\sigma[..j]\tau \in \bigcap_{i \in I} P_i$. We have to show that $\sigma \in \bigcap_{i \in I} P_i$. By the assumption, for every $i \in I$ and every $j \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $\sigma[..j]\tau \in P_i$, implying $\sigma \in P_i$ for every $i \in I$ because all P_i were assumed to be safety properties. Hence, $\sigma \in \bigcap_{i \in I} P_i$.

(ii) Again, the assertion is trivial for $n = 0$ since $\emptyset$ is a safety property. Else, assume $\sigma \notin \bigcup_{i=1}^{n} P_i$ for some $\sigma \in \Sigma^\omega$. Thus, $\sigma \notin P_i$ for every $1 \leq i \leq n$ and, since every P_i is a safety property, there exist $m_1, \ldots, m_n$ such that for every $\tau \in \Sigma^\omega$, $\sigma[..m_i]\tau \notin P_i$ for $1 \leq i \leq n$. In particular, for $m := \max\{m_1, \ldots, m_n\}$, $\sigma[..m]\tau \notin P_i$ for every $\tau \in \Sigma^\omega$ and $1 \leq i \leq n$. Hence, $\sigma[..m]\tau \notin \bigcup_{i=1}^{n} P_i$ for every $\tau \in \Sigma^\omega$, which proves that $\bigcup_{i=1}^{n} P_i$ is a safety property.

♠

Notice that in general only finite unions of safety properties are themselves safety properties. An example for an infinite set of safety properties whose union is not a safety property will be given following theorem 2.30.

For a liveness property P, it is obvious from the definition that every property $Q \supseteq P$ also is a liveness property. In particular, arbitrary unions of liveness properties yield liveness properties. In general, however, even finite intersections of liveness properties are not liveness properties. For example, consider $P = \{\sigma \in \Sigma^\omega \mid \exists m \geq 0 \forall n \geq m\ \sigma[n] = s\}$ and $Q = \{\sigma \in \Sigma^\omega \mid \forall m \geq 0 \exists n \geq m\ \sigma[n] \neq s\}$ where $s \in \Sigma$ is some designated state. It is easy to verify that both P and Q are liveness properties (assuming that Σ contains at least two different states) whereas $P \cap Q = \emptyset$ which is certainly not a liveness property.

The classes of safety and liveness properties are "almost" disjoint:

Lemma 2.3 *Σ^ω is the only property which is both a safety and a liveness property.*

Proof: Clearly, Σ^ω satisfies both the requirements of safety and liveness properties. Assume $P \subseteq \Sigma^\omega$ is both a safety property and a liveness property and let $\sigma \in \Sigma^\omega$ be arbitrary. Then for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $\sigma[..i]\tau \in P$ since P is a liveness property. Since P is a safety property, this implies $\sigma \in P$, hence $P = \Sigma^\omega$. ♠

In particular, thus, for every property P there is a safety property $S \supseteq P$. By lemma 2.2, there is a smallest safety property (w.r.t. set inclusion) containing P, viz. the intersection of all such safety properties.

Definition 2.4 (safety closure) *Let P be any property. The* safety closure *$safe(P)$ of P is the smallest safety property containing P,*

$$safe(P) := \bigcap\{S \subseteq \Sigma^\omega \mid S \textit{ is a safety property and } P \subseteq S\}$$

This abstract definition of the safety closure of P is not very useful for practical purposes. There is, however, a more constructive characterization of this notion in terms of the set of prefixes of P:

Let P be a property. Then $\mathit{Pref}(P)$ denotes the set of all prefixes of P:

$$\mathit{Pref}(P) = \{\sigma \in \Sigma^* \mid \exists \sigma' \in P : \sigma \prec \sigma'\}.$$

Theorem 2.5 *Let P be a property and let $\mathit{scl}(P)$ denote the set*

$$\mathit{scl}(P) = \{\sigma \in \Sigma^\omega \mid \forall \sigma' \in \Sigma^+ : \sigma' \prec \sigma \Rightarrow \sigma' \in \mathit{Pref}(P)\}.$$

Then $\mathit{scl}(P) = \mathit{safe}(P)$.

Proof:

- $P \subseteq \mathit{scl}(P)$: Let $\sigma \in P$. Obviously, for every $\sigma' \prec \sigma$, $\sigma' \in \mathit{Pref}(P)$, hence $\sigma \in \mathit{scl}(P)$.

- $\mathit{scl}(P)$ is a safety property: Assume $\sigma \in \Sigma^\omega$ such that $\forall i \geq 0 \exists \tau \in \Sigma^\omega(\sigma[..i]\tau \in \mathit{scl}(P))$. We have to prove $\sigma \in \mathit{scl}(P)$. Let $\sigma' \prec \sigma$ be an arbitrary non-empty prefix of σ. Then $\sigma' = \sigma[..i]$ for some $i \geq 0$, thus there is some $\tau \in \Sigma^\omega$ such that $\sigma'\tau \in \mathit{scl}(P)$. By $\sigma' \prec \sigma'\tau$ and the definition of $\mathit{scl}(P)$, $\sigma' \in \mathit{Pref}(P)$ and, since $\sigma' \prec \sigma$ was arbitrary, $\sigma \in \mathit{scl}(P)$, again by the definition of $\mathit{scl}(P)$.

- $\mathit{scl}(P)$ is the smallest safety property containing P: Let S be an arbitrary safety property such that $P \subseteq S$. We have to show $\mathit{scl}(P) \subseteq S$. Assume $\sigma \notin S$ for some $\sigma \in \Sigma^\omega$. Since S is a safety property, there is some $i \geq 0$ such that for every $\tau \in \Sigma^\omega$, $\sigma[..i]\tau \notin S$. Since $P \subseteq S$, it follows that $\sigma[..i]\tau \notin P$ for every $\tau \in \Sigma^\omega$. But then there is some prefix $\sigma' \prec \sigma$ (viz., $\sigma' = \sigma[..i]$) such that $\sigma' \notin \mathit{Pref}(P)$, hence $\sigma \notin \mathit{scl}(P)$. ♠

Example 2.6 Let $A, B \subseteq \Sigma$ and let U be the property defined by

$$\begin{array}{rcl} U &=& \{\sigma \in \Sigma^\omega \mid \exists i \geq 0\ (\sigma[i] \in B \text{ and } \forall j < i\ \sigma[j] \in A)\}. \\ \text{Then, } \mathit{Pref}(U) &=& \{\sigma \in \Sigma^* \mid \exists 0 \leq i < |\sigma|\ (\sigma[i] \in B \text{ and } \forall j < i\ \sigma[j] \in A)\} \cup \\ && \{\sigma \in \Sigma^* \mid \forall 0 \leq i < |\sigma|\ (\sigma[i] \in A)\} \\ \text{and } \mathit{scl}(U) &=& \{\sigma \in \Sigma^\omega \mid \sigma \in U \text{ or } \forall i \geq 0\ (\sigma[i] \in A)\}. \end{array}$$

Notice that in temporal logic the property U would be characterized by the formula A until B whereas $\mathit{scl}(U)$ corresponds to A unless B. ♠

Having defined the notion of the safety closure of some property P, there is also a liveness property which is naturally associated with P:

Definition 2.7 (liveness extension) *Let P be any property. Then*

$$live(P) := P \cup (\Sigma^\omega \setminus safe(P))$$

is called the liveness extension *of P.*

Lemma 2.8 *For any property P, $live(P)$ is a liveness property. Moreover, $safe(P) \cap live(P) = P$, and $safe(P) \cup live(P) = \Sigma^\omega$.*

Proof: To prove that $live(P)$ is indeed a liveness property, let $\sigma \in \Sigma^*$ be any finite state sequence. If $\sigma \in Pref(P)$, then by definition there is some $\tau \in \Sigma^\omega$ such that $\sigma\tau \in P \subseteq live(P)$. If $\sigma \notin Pref(P)$, then for every $\tau \in \Sigma^\omega$, $\sigma\tau \notin scl(P)$ (by the definition of $scl(P)$, using $\sigma \circ \tau[0] \prec \sigma\tau$). In particular, there is some $\tau \in \Sigma^\omega$ such that $\sigma\tau \notin scl(P) = safe(P)$, but this implies $\sigma\tau \in live(P)$.

The asserted equations follow from simple calculations:

$safe(P) \cap live(P) = safe(P) \cap (P \cup (\Sigma^\omega \setminus safe(P)))$
$= (safe(P) \cap P) \cup (safe(P) \cap (\Sigma^\omega \setminus safe(P))) = P \cup \emptyset = P$,

$safe(P) \cup live(P) = safe(P) \cup (P \cup (\Sigma^\omega \setminus safe(P))) = P \cup \Sigma^\omega = \Sigma^\omega$ ♠

In particular, the preceding lemma immediately yields the following corollary:

Corollary 2.9 *Let P be any property. Then there exist a safety property S and a liveness property L such that $P = S \cap L$.*

Notice that, although $live(P)$ is one example for a liveness property L satisfying $safe(P) \cap L = P$, it is not necessarily the smallest such liveness property. Consider, for instance, the property U defined in example 2.6 above:

$$\begin{aligned} live(U) &= U \cup (\Sigma^\omega \setminus safe(U)) \\ &= U \cup \{\sigma \in \Sigma^\omega \mid \sigma \notin U \text{ and } \exists i \geq 0\ \sigma[i] \notin A\} \\ &= \{\sigma \in \Sigma^\omega \mid \exists i \geq 0\ (\sigma[i] \in B \text{ and } \forall j < i\ \sigma[j] \in A) \text{ or} \\ &\qquad\qquad \exists i \geq 0\ \sigma[i] \notin A\} \\ &= \{\sigma \in \Sigma^\omega \mid \exists i \geq 0\ \sigma[i] \in B \text{ or } \exists i \geq 0\ \sigma[i] \notin A\} \end{aligned}$$

whereas the property $L \subseteq live(U)$ — the inclusion being strict if there is at least one state $s \in \Sigma \setminus (A \cup B)$ — defined by

$$L = \{\sigma \in \Sigma^\omega \mid \exists i \geq 0\ \sigma[i] \in B\}$$

also satisfies $L \cap safe(U) = U$. In fact, L corresponds to the temporal logic formula $\Diamond B$, and A `until` $B \leftrightarrow (A$ `unless` $B) \wedge \Diamond B$ is a valid equivalence in temporal logic. However, for an arbitrary property P no smallest liveness property containing P need exist since intersections of liveness properties do not generally yield another liveness property (see the remark following lemma 2.2). The above definition of $live(P)$, on the other hand, is simple to define and suffices for our purposes.

Using the notions of safety closure and liveness extension, new characterizations of safety and liveness properties can be obtained:

Theorem 2.10

- *P is a safety property iff $safe(P) = P$ iff $live(P) = \Sigma^\omega$.*
- *P is a liveness property iff $safe(P) = \Sigma^\omega$ iff $live(P) = P$.*

Proof: P is a safety property

$$
\begin{array}{ll}
\Longleftrightarrow & \forall\sigma \in \Sigma^\omega\ (\sigma \in P \ \Leftrightarrow\ \forall i \geq 0 \exists \tau \in \Sigma^\omega\ \sigma[..i]\tau \in P) \\
\Longleftrightarrow & \forall\sigma \in \Sigma^\omega(\sigma \in P \ \Leftrightarrow\ \forall \sigma' \in \Sigma^+\ \sigma' \prec \sigma \ \Rightarrow\ \sigma' \in Pref(P)) \\
\Longleftrightarrow & \forall\sigma \in \Sigma^\omega(\sigma \in P \ \Leftrightarrow\ \sigma \in scl(P)) \\
\Longleftrightarrow & P = safe(P) \quad \text{(by theorem 2.5)} \\
\Longleftrightarrow & P \supseteq safe(P) \quad \text{(since } P \subseteq safe(P) \text{ follows from def. 2.4)} \\
\Longleftrightarrow & live(P) = P \cup (\Sigma^\omega \setminus safe(P)) = \Sigma^\omega
\end{array}
$$

P is a liveness property

$$
\begin{array}{ll}
\Longleftrightarrow & \forall\sigma \in \Sigma^* \exists \tau \in \Sigma^\omega\ \sigma\tau \in P \\
\Longleftrightarrow & \forall\sigma \in \Sigma^+\ \sigma \in Pref(P) \\
\Longleftrightarrow & safe(P) = scl(P) = \Sigma^\omega \\
\Longleftrightarrow & \Sigma^\omega \setminus safe(P) \subseteq P \quad (\Leftarrow \text{by } P \subseteq safe(P)) \\
\Longleftrightarrow & live(P) = P \cup (\Sigma^\omega \setminus safe(P)) = P
\end{array}
$$

♠

We conclude this section by stating, for future reference, a simple but fundamental lemma concerning the natural consequence relation which can be defined for properties by *Q follows from P* ($P \models Q$) iff $P \subseteq Q$.

Lemma 2.11 *Let Q be a safety property and $P \subseteq \Sigma^\omega$ be arbitrary. Then*

$$P \models Q \quad \Longleftrightarrow \quad safe(P) \models Q.$$

Proof:
$\Longrightarrow$: Let $P \subseteq Q$ where Q is a safety property. By definition, $safe(P)$ is the smallest safety property S such that $P \subseteq S$, hence $safe(P) \subseteq Q$.

$\Longleftarrow$: For any property P, it holds that $P \subseteq safe(P)$. Assuming $safe(P) \subseteq Q$, $P \subseteq Q$ follows. ♠

2.2 Propositional νTL

The previous section dealt with a very general definition of properties as arbitrary subsets of infinite state sequences. For practical purposes, especially for program development or verification, it is important to develop a concise notation equipped with a supporting "calculus" of reasoning rules. Temporal logics, combining a classical logical formalism for the reasoning about single states with additional operators relating several states of runs, have been used successfully in this field. Of the great number of pertinent publications, let us only mention Kröger [29] and Manna & Pnueli [36]. As for the choice of basic temporal operators, we will simply allow every connective whose semantics can be characterized as a maximal or minimal solution of a system of fixpoint equations. In this way, an expressive power is attained which encompasses that of other, more conventional, temporal logics (see Wolper [59] and Stirling [47]) without sacrificing desirable proof-theoretic properties of the formalism, such as a reasonably natural axiomatization or decision procedures (for the propositional case — see chapter 3).

Syntax of propositional νTL

We begin with defining syntax and semantics of the propositional version of the linear time temporal logic νTL introduced by Banieqbal & Barringer [10]. The logic νTL is based on the ν-calculus which was notably studied by Kozen [26], cf. also Stirling [47].

Let $\mathcal{V}$ be a denumerable non-empty set of *proposition letters*, typical representatives being $p, q, v, v_1, v_2, \ldots$ and $\mathcal{X}$ be a denumerable set of *proposition variables* $X, Y, X_1, X_2, \ldots$. The *alphabet* of a language $\mathcal{L}$ of propositional νTL contains $\mathcal{V}$, $\mathcal{X}$, the symbols $\neg$, $\wedge$, $\circ$, ν, the dot ., and parentheses.

Formulas of $\mathcal{L}$ are built according to the following inductive definition:

- Every proposition letter $v \in \mathcal{V}$ is a formula.
- Every proposition variable $X \in \mathcal{X}$ is a formula.

- If F is a formula, then $\neg F$ and $\circ F$ are formulas.
- If F and G are formulas, then $(F \wedge G)$ is a formula.
- If F is a formula and $X \in \mathcal{X}$ is a proposition variable such that all occurrences of X in F except for those occurring in a subformula $\nu X.G$ of F are *positive* (i.e., they are in the scope of an even number of negation symbols), then $(\nu X.F)$ is a formula.

The occurrence of a proposition variable X in some formula F is called *bound* if it appears in some subformula $(\nu X.G)$ of F. Otherwise, it is called *free*. Formulas F containing no free (occurrences of) proposition variables are called *closed formulas*.

If F and G are formulas and X is a proposition variable, $F[G/X]$ denotes the formula resulting from F by replacing all free occurrences of the variable X in F by the formula G. Hereby we tacitly assume that no proposition variable which occurs free in G becomes bound in $F[G/X]$. If this situation would result from a mere textual replacement of X by G, some bound variables in F will be (consistently) renamed before the substitution. Notice that if all occurences of X in F are positive, then the polarity of all free variable occurrences in G will be preserved. It will be reversed if all occurrences of X in F are negative.

Further logical connectives as well as the constants t and f are defined as syntactical abbreviations as follows:

$$
\begin{array}{rcl}
(F \vee G) & \equiv & \neg(\neg F \wedge \neg G) \\
(F \rightarrow G) & \equiv & (\neg F \vee G) \\
(F \leftrightarrow G) & \equiv & ((F \rightarrow G) \wedge (G \rightarrow F)) \\
\circ^i F & \equiv & \left\{ \begin{array}{ll} F & \text{if } i = 0 \\ \circ\circ^{i-1} F & \text{else} \end{array} \right\} \text{ for } i \in I\!N \\
(\mu X.F) & \equiv & \neg(\nu X.\neg F[\neg X/X]), \\
 & & \text{if all free occurrences of } X \text{ in } F \text{ are positive} \\
\mathsf{t} & \equiv & (v \rightarrow v) \text{ for some } v \in \mathcal{V} \\
\mathsf{f} & \equiv & \neg \mathsf{t}
\end{array}
$$

Notice that all free occurrences of X in $\neg F[\neg X/X]$ are positive if this condition holds for F. Thus, $(\mu X.F)$ abbreviates a well-formed formula.

To avoid writing too many parentheses, we define the following priority order among operators (listing them in order of decreasing priority): $\neg$, $\circ$, $\wedge$, $\vee$, $\rightarrow$, ν, μ, $\leftrightarrow$. Also, outermost parentheses will be omitted.

Semantics of propositional νTL

A *temporal structure* $\mathcal{K}$ for propositional νTL consists of an ω-sequence $\sigma = (\sigma_0, \sigma_1, \ldots)$ of *states* where each state is a mapping $\sigma_i : \mathcal{V} \to 2^{\{()\}}$. (Intuitively, the empty set represents the truth value "false", and the set $\{()\}$ consisting of the empty tuple represents "true". This somewhat cumbersome notation allows to subsume the semantics of propositional νTL by the following semantic definitions for the first-order version of νTL.)

The *semantics* of propositional νTL is given by a temporal structure $\mathcal{K}$ together with a valuation $\xi : \mathcal{X} \to 2^{\mathbb{N}}$ of the proposition variables. If $I \subseteq \mathbb{N}$ is a set of natural numbers, ξ is a valuation of the proposition variables and $X \in \mathcal{X}$, $\xi[I/X]$ denotes the valuation defined by $\xi[I/X](Y) = \xi(Y)$ for every variable $Y \neq X$, and $\xi[I/X](X) = I$.

Given a temporal structure $\mathcal{K}$ and a valuation ξ, for every formula F the set $\mathcal{K}^\xi(F) \subseteq \mathbb{N}$ is inductively defined as follows:

$$\begin{array}{rcl}
\mathcal{K}^\xi(v) &=& \{i \in \mathbb{N} \mid \sigma_i(v) = \{()\}\} \quad \text{for every proposition letter } v \in \mathcal{V}, \\
\mathcal{K}^\xi(X) &=& \xi(X) \quad \text{for every proposition variable } X \in \mathcal{X}, \\
\mathcal{K}^\xi(\neg F) &=& \mathbb{N} \setminus \mathcal{K}^\xi(F), \\
\mathcal{K}^\xi(F \wedge G) &=& \mathcal{K}^\xi(F) \cap \mathcal{K}^\xi(G), \\
\mathcal{K}^\xi(\circ F) &=& \{i \in \mathbb{N} \mid i+1 \in \mathcal{K}^\xi(F)\}, \\
\mathcal{K}^\xi(\nu X.F) &=& \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \mathcal{K}^{\xi[I/X]}(F)\}.
\end{array}$$

It follows (see page 23) that the semantics of the μ-operator is given by

$$\mathcal{K}^\xi(\mu X.F) \;=\; \bigcap\{I \subseteq \mathbb{N} \mid I \supseteq \mathcal{K}^{\xi[I/X]}(F)\}.$$

Intuitively, $\mathcal{K}^\xi(F)$ is the set of (indexes of) states where the formula F is true in the structure $\mathcal{K}$ under the valuation ξ.

Example 2.12 Let us explicitly calculate $\mathcal{K}^\xi(F)$ for the formula

$$F \equiv \nu X.\neg p \to q \wedge \circ X,$$

a temporal structure $\mathcal{K}$ whose state sequence σ assigns $\sigma_i(p) = \{()\}$ iff $i \in \{3, 6, 9\}$ and $\sigma_i(q) = \{()\}$ iff $i \geq 7$, and an arbitrary valuation ξ of the proposition variables.

Consider, at first, the set $I = \{i \in \mathbb{N} \mid i = 3 \text{ or } i \geq 6\}$:

$$
\begin{aligned}
& \mathcal{K}^{\xi[I/X]}(\neg p \rightarrow q \wedge \circ X) \\
= \ & \mathcal{K}^{\xi[I/X]}(p) \cup (\mathcal{K}^{\xi[I/X]}(q) \cap \mathcal{K}^{\xi[I/X]}(\circ X)) \\
= \ & \{3,6,9\} \cup (\{i \in I\!N \mid i \geq 7\} \cap \{i \in I\!N \mid i+1 \in I\}) \\
= \ & \{3,6,9\} \cup (\{i \in I\!N \mid i \geq 7\} \cap \{i \in I\!N \mid i = 2 \text{ or } i \geq 5\}) \\
= \ & \{3,6,9\} \cup \{i \in I\!N \mid i \geq 7\} \\
= \ & \{i \in I\!N \mid i = 3 \text{ or } i \geq 6\} \ = \ I
\end{aligned}
$$

In particular, $I \subseteq \mathcal{K}^{\xi[I/X]}(\neg p \rightarrow q \wedge \circ X)$. On the other hand, let $I \subseteq I\!N$ be an arbitrary set and assume that $0 \in I$. Since $0 \notin \mathcal{K}^{\xi[I/X]}(p)$ and $0 \notin \mathcal{K}^{\xi[I/X]}(q)$, it follows that $0 \notin \mathcal{K}^{\xi[I/X]}(\neg p \rightarrow q \wedge \circ X)$, implying $I \not\subseteq \mathcal{K}^{\xi[I/X]}(\neg p \rightarrow q \wedge \circ X)$. Similar arguments hold for $i = 1,2,4,5$. It follows that $\mathcal{K}^{\xi}(F) = \{i \in I\!N \mid i = 3 \text{ or } i \geq 6\}$. ♠

2.3 First-order νTL

Syntax of first-order νTL

The alphabet of a language $\mathcal{L}$ of first-order νTL contains

- a denumerable set $\mathcal{Y} = \{x, y, x_0, \ldots\}$ of *rigid subject variables*,
- a denumerable set $\mathcal{Z} = \{z, z_0, z_1, \ldots\}$ of *flexible subject variables*,
- for every $n \geq 0$, at most denumerable sets of n-ary *function symbols* $f, g, h, \ldots$ (called *constants* $c, c_0, \ldots$ for $n = 0$),
- for every $n \geq 0$, at most denumerable sets of n-ary *predicate symbols* $P, Q, R, \ldots$ (called *proposition letters* for $n = 0$),
- the binary predicate symbol $=$,
- a denumerable set $\mathcal{X}$ of *proposition variables*,
- the symbols $\neg, \wedge, \circ, \nu, \forall$, the dot and parentheses.

Terms of $\mathcal{L}$ are inductively defined from rigid and flexible subject variables by applying function symbols as usual. The sets of rigid and flexible subject variables occurring in a term t are denoted by $FRV(t)$ and $FFV(t)$, respectively.

Atomic formulas of $\mathcal{L}$ are either of the form $P(t_1, \ldots, t_n)$ where P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, or $t_1 = t_2$ for two terms t_1 and t_2, or a proposition variable $X \in \mathcal{X}$. For proposition letters, we will write P instead of $P()$.

Formulas of $\mathcal{L}$ are defined from atomic formulas similarly as in the propositional case from proposition letters and variables. The only additional clause is: If F is a formula which does not contain free occurrences of any proposition variable and $y \in \mathcal{Y}$ is a rigid variable then $\forall y F$ is a formula. This restriction avoids free proposition variables in the scope of first-order quantifiers which might lead to undesired interferences between propositional and first-order quantification.

State formulas are formulas which do not contain any proposition variables nor the temporal operators $\circ$ or ν.

Free and bound occurrences of (rigid) subject variables are defined as usual; for proposition variables these notions have already been introduced in the section on propositional νTL. By $FFV(F)$ and $FRV(F)$, respectively, we denote the sets of flexible and free rigid subject variables occurring in the formula F. Notice that no quantification over flexible subject variables is allowed. A formula F containing no free occurrences of rigid variables is called *quantifier closed*. If F is a closed formula (i.e., without free occurrences of proposition variables) and $\{x_1, \ldots, x_n\}$ is the set of rigid subject variables occurring freely in F, the *universal closure* of F, denoted by $(\forall *)F$ is defined as the formula $\forall x_1 \ldots \forall x_n F$.

$F[t/y]$ denotes the formula resulting from F by replacing all free occurrences of the rigid variable y in F by the term t where again bound variables in F will be implicitly renamed such that no variable occurring in t becomes bound in $F[t/y]$. The analogous notation $F[G/X]$ denoting the replacement of the free proposition variable X by a formula G has already been introduced in the section on propositional νTL.

Further logical connectives are defined as in the propositional case with the additional definition of $\exists y F \equiv \neg\forall y \neg F$ (implying the condition that F does not contain any free proposition variable). Besides, we will write $t_1 \neq t_2$ instead of $\neg(t_1 = t_2)$. The operator precedence is as above with quantifiers $\forall, \exists$ binding strongest.

Semantics of first-order νTL

Let $\mathcal{L}$ be a first-order νTL-language. A *first-order structure* $\mathcal{S}$ for $\mathcal{L}$ consists of a non-empty set $|\mathcal{S}|$, the *universe* of $\mathcal{S}$ and, for each n-ary function symbol f in $\mathcal{L}$, a function $\mathcal{S}(f) : |\mathcal{S}|^n \to |\mathcal{S}|$.

A *temporal structure* for $\mathcal{L}$ is a pair $\mathcal{K} = (\mathcal{S}, \sigma)$ where $\mathcal{S}$ is a first-order structure for $\mathcal{L}$ and $\sigma = (\sigma_0, \sigma_1, \ldots)$ is an ω-sequence of states. Every state σ_i assigns some value $\sigma_i(z) \in |\mathcal{S}|$ to each flexible variable $z \in \mathcal{Z}$ and some subset $\sigma_i(P) \subseteq$

$|\mathcal{S}|^n$ (where $|\mathcal{S}|^0 := \{()\}$ is the set consisting of the empty tuple) to each n-ary predicate symbol P of $\mathcal{L}$.

For every term t, the temporal structure $\mathcal{K}$ together with a valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ of the rigid variables defines an ω-sequence $\mathcal{K}^\eta(t) = (\mathcal{K}^\eta_0(t), \mathcal{K}^\eta_1(t), \ldots)$ of values $\mathcal{K}^\eta_i(t) \in |\mathcal{S}|$ by

$$\begin{array}{rcll} \mathcal{K}^\eta_i(y) & = & \eta(y) & \text{for rigid subject variables } y \in \mathcal{Y} \\ \mathcal{K}^\eta_i(z) & = & \sigma_i(z) & \text{for flexible subject variables } z \in \mathcal{Z} \\ \mathcal{K}^\eta_i(f(t_1, \ldots, t_n)) & = & \mathcal{S}(f)(\mathcal{K}^\eta_i(t_1), \ldots, \mathcal{K}^\eta_i(t_n)) & \text{otherwise.} \end{array}$$

Notice that $\mathcal{K}^\eta_i(t) = \mathcal{K}^\eta_j(t)$ for all $i, j \in \mathbb{N}$ if t does not contain flexible variables.

A temporal structure $\mathcal{K}$ together with valuations $\eta : \mathcal{Y} \to |\mathcal{S}|$ and $\xi : \mathcal{X} \to 2^{\mathbb{N}}$ of the rigid subject variables and the proposition variables of $\mathcal{L}$, respectively, assigns a set $\mathcal{K}^{\eta,\xi}(F) \subseteq \mathbb{N}$ to every formula F in a way similar to the corresponding definition in the propositional case. We only give the new cases (for atomic formulas F and for the quantifier step):

$$\begin{array}{rcl} \mathcal{K}^{\eta,\xi}(P(t_1, \ldots, t_n)) & = & \{i \in \mathbb{N} \mid (\mathcal{K}^\eta_i(t_1), \ldots, \mathcal{K}^\eta_i(t_n)) \in \sigma_i(P)\} \\ \mathcal{K}^{\eta,\xi}(t_1 = t_2)) & = & \{i \in \mathbb{N} \mid \mathcal{K}^\eta_i(t_1) = \mathcal{K}^\eta_i(t_2)\} \\ \mathcal{K}^{\eta,\xi}(\forall y F) & = & \bigcap_{d \in |\mathcal{S}|} \mathcal{K}^{\eta[d/y],\xi}(F) \end{array}$$

(where $\eta[d/y](y) = d$ and $\eta[d/y](y') = \eta(y')$ for every rigid variable $y' \neq y$).

Observe that propositional νTL can be seen as just a special case of first-order νTL for a restricted language whose only predicate symbols are proposition letters (and where, consequently, no terms need to be considered). In the sequel, unless stated explicitly, νTL refers to first-order νTL.

We say that a formula F is *true at state σ_i under the valuations η and ξ* if $i \in \mathcal{K}^{\eta,\xi}(F)$. This will be denoted by $\mathcal{K}^{\eta,\xi}_i \models F$. F is called *true in the structure $\mathcal{K}$ under η and ξ* ($\mathcal{K}^{\eta,\xi} \models F$) if $\mathcal{K}^{\eta,\xi}_0 \models F$. A formula F is called *satisfiable* if there are $\mathcal{K}$, η and ξ such that $\mathcal{K}^{\eta,\xi} \models F$. F is called *valid* (denoted by $\models F$) if $\mathcal{K}^{\eta,\xi} \models F$ holds for every structure $\mathcal{K}$ under all valuations η and ξ. If F is a closed (respectively, quantifier closed) formula, the definition of $\mathcal{K}^{\eta,\xi}(F)$ is independent of ξ (of η, resp.), and in this case ξ or η will frequently be omitted. In particular, if F is a closed and quantifier closed formula, a temporal structure $\mathcal{K}$ such that $\mathcal{K} \models F$ is called a *model* of F. $Mod(F)$ denotes the set of models of F. A closed and quantifier closed formula F *follows from* a set $\mathcal{F}$ of such formulas (denoted by $\mathcal{F} \models F$) if $\mathcal{K} \models F$ holds for every temporal structure $\mathcal{K}$ which is a model of every formula $G \in \mathcal{F}$. Two formulas F and G are called *equivalent* if $\mathcal{K}^{\eta,\xi}(F) = \mathcal{K}^{\eta,\xi}(G)$ for every temporal structure $\mathcal{K}$ and all valuations η and ξ.

The fixpoint formulas $\nu X.F$ and $\mu X.F$ are well-formed formulas only if every free occurrence of X in F is in the scope of an even number of negation symbols. The following monotonicity and fixed point properties of the semantics of such formulas are based on this restriction.

Lemma 2.13 *Let $\mathcal{K}$ be a temporal structure and $I \subseteq J \subseteq I\!N$ be two sets. Then, for all valuations η and ξ of the rigid subject and the proposition variables, respectively, $\mathcal{K}^{\eta,\xi[I/X]}(F) \subseteq \mathcal{K}^{\eta,\xi[J/X]}(F)$ (resp., $\mathcal{K}^{\eta,\xi[I/X]}(F) \supseteq \mathcal{K}^{\eta,\xi[J/X]}(F)$) if all free occurrences of X appear under an even (resp., odd) number of negation symbols.*

Proof: The proof proceeds by induction on the structure of the formula F. If F is an atomic formula $P(t_1, \ldots, t_n)$ or $t_1 = t_2$, or if $F \equiv Y \in \mathcal{X}$, where $Y \neq X$, obviously $\mathcal{K}^{\eta,\xi[I/X]}(F) = \mathcal{K}^{\eta,\xi[J/X]}(F)$. For $F \equiv X$, $\mathcal{K}^{\eta,\xi[I/X]}(F) = I \subseteq J = \mathcal{K}^{\eta,\xi[J/X]}(F)$, as required. For the induction step, we will assume that all free occurrences of X in F are positive, the other case being symmetrical. Let $F \equiv \neg G$. Then all free occurrences of X in G are negative, and by the induction hypothesis, $\mathcal{K}^{\eta,\xi[I/X]}(G) \supseteq \mathcal{K}^{\eta,\xi[J/X]}(G)$, hence $\mathcal{K}^{\eta,\xi[I/X]}(F) = I\!N \setminus \mathcal{K}^{\eta,\xi[I/X]}(G) \subseteq I\!N \setminus \mathcal{K}^{\eta,\xi[J/X]}(G) = \mathcal{K}^{\eta,\xi[J/X]}(F)$. The proofs for the cases $F \equiv G \wedge H$, $F \equiv \circ G$ and $F \equiv \forall y G$ are similar. Assume $F \equiv \nu Y.G$. If $Y \equiv X$, $\mathcal{K}^{\eta,\xi[I/X]}(F) = \mathcal{K}^{\eta,\xi[J/X]}(F)$ since F does not contain free occurrences of X. Otherwise, X appears only positively in G, and by the induction hypothesis $\mathcal{K}^{\eta,\xi'[I/X]}(G) \subseteq \mathcal{K}^{\eta,\xi'[J/X]}(G)$ for every valuation ξ' of the proposition variables. Assume $N \subseteq \mathcal{K}^{\eta,\xi[I/X][N/Y]}(G)$ for a set $N \subseteq I\!N$. Since $Y \neq X$, $\xi[I/X][N/Y] = \xi[N/Y][I/X]$ (and similarly for J), hence $N \subseteq \mathcal{K}^{\eta,\xi[J/X][N/Y]}(G)$ (using the induction hypothesis) and therefore $N \subseteq \mathcal{K}^{\eta,\xi[J/X]}(\nu X.G)$. Taking the union over all such sets N yields the claim of the lemma. ♠

In particular, for every formula $\nu X.F$ or $\mu X.F$ (and fixed temporal structure $\mathcal{K}$ and valuations ξ and η of the proposition variables and the rigid subject variables) the mapping $\tau_F : 2^{I\!N} \to 2^{I\!N}, I \mapsto \mathcal{K}^{\eta,\xi[I/X]}(F)$ is monotonic on the complete lattice $(2^{I\!N}, \subseteq)$ since all free occurrences of X in F are positive. The fixpoint theorem of Knaster and Tarski (Tarski [53]) therefore ensures the existence of both a smallest fixpoint $\mu_F \subseteq I\!N$ and a greatest fixpoint ν_F of τ_F given by

$$
\begin{aligned}
\mu_F &= \bigcap\{I \subseteq I\!N \mid \tau_F(I) \subseteq I\} &= \bigcap\{I \subseteq I\!N \mid \tau_F(I) = I\} \\
\nu_F &= \bigcup\{I \subseteq I\!N \mid I \subseteq \tau_F(I)\} &= \bigcup\{I \subseteq I\!N \mid I = \tau_F(I)\},
\end{aligned}
$$

respectively. Clearly, $\mu_F = \mathcal{K}^{\eta,\xi}(\mu X.F)$ and $\nu_F = \mathcal{K}^{\eta,\xi}(\nu X.F)$. Due to this characterization of the semantics of formulas $\nu X.F$ and $\mu X.F$, such formulas will sometimes be referred to as *fixpoint formulas*. The alternative representations of their semantics provided by the Knaster-Tarski theorem where $=$ replaces $\subseteq$ will subsequently be used whenever they seem more appropriate.

A natural question arising in this context concerns the continuity of the maps τ_F. Academic interest aside, continuous mappings are of practical importance because their fixpoints can be "computed" in an ω-iteration which means that they can be approximated effectively to any desired degree of precision. (For example, PROLOG interpreters rely on this fact: The minimal Herbrand model of definite Horn clause programs can be characterized as the fixpoint of a continuous mapping on the lattice of Herbrand pre-interpretations which can be approximated by an ω-iteration. This fact is a crucial step in the proof of completeness of SLD-resolution, see Lloyd [35].)

Let $\mathcal{I} = \{I_j \mid j \in J\}$ be a non-empty set of subsets $I_j \subseteq I\!N$. $\mathcal{I}$ is a *directed set* if for every two sets $I_j, I_k \in \mathcal{I}$ there is some $I_m \in \mathcal{I}$ such that $I_j \subseteq I_m$ and $I_k \subseteq I_m$. The mapping τ_F is *continuous* if, for every directed set $\mathcal{I} = \{I_j \mid j \in J\}$,

$$\bigcup_{j \in J} \tau_F(I_j) = \tau_F(\bigcup_{j \in J} I_j).$$

For a continuous mapping τ_F, the Kleene fixpoint theorem gives simpler characterizations of its smallest and greatest fixpoints, viz.

$$\mu_F = \bigcup_{i=0}^{\omega} \tau_F^i(\emptyset) \quad \text{and} \quad \nu_F = \bigcap_{i=0}^{\omega} \tau_F^i(I\!N),$$

respectively (where $\tau_F^i(I)$ denotes the i-fold iteration of τ_F applied to I).

Unfortunately, there are νTL formulas F which induce non-continuous mappings τ_F for certain temporal structures. A propositional example is provided by the formula

$$F \equiv \nu X.((p \vee (q \wedge \circ X)) \wedge \circ(\mu Y.X \vee \circ Y)).$$

Let $\mathcal{K}$ be the temporal structure where $\sigma_i(p) = \{()\}$ iff $i = 0$ and $\sigma_i(q) = \emptyset$ iff $i = k^2$ for some $k \in I\!N$. Then

$$\begin{array}{rll} & \tau_F(I) = & (\{0\} \cup \tau_1(I)) \cap \tau_2(I) \\ \text{where} & \tau_1(I) = \mathcal{K}^{\xi[I/X]}(q \wedge \circ X) & = \{i \in I\!N \mid i \neq k^2 \text{ and } i+1 \in I\} \\ \text{and} & \tau_2(I) = \mathcal{K}^{\xi[I/X]}(\circ(\mu Y.X \vee \circ Y)) & = \{i \in I\!N \mid \exists j > i\ (j \in I)\}. \end{array}$$

The following table shows a calculation of $\tau_F^j(I\!N)$ for $j = 0, 1, \ldots$:

j	$\tau_F^j(I\!N)$	$\tau_1(\tau_F^j(I\!N))$	$\tau_2(\tau_F^j(I\!N))$
0	$I\!N$	$\{i \mid i \neq k^2\}$	$I\!N$
1	$\{i \mid i \neq k^2\} \cup \{0\}$	$\{i \mid i \neq k^2, i \neq k^2 - 1\}$	$I\!N$
2	$\{i \mid i \neq k^2, i \neq k^2 - 1\} \cup \{0\}$	$\{i \mid i \neq k^2, i \neq k^2 - 1,$ $i \neq k^2 - 2\}$	$I\!N$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
j	$\{i \mid i \neq k^2, i \neq k^2 - 1, \ldots,$ $i \neq k^2 - j + 1\} \cup \{0\}$	$\{i \mid i \neq k^2, i \neq k^2 - 1, \ldots,$ $i \neq k^2 - j\}$	$I\!N$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

The point is that for every $i \in I\!N$ and every $j \geq 0$ there is some $l > j$ such that $l \in \tau_F^j(I\!N)$ forcing $\tau_2(\tau_F^j(I\!N)) = I\!N$ for every $j \geq 0$. In particular, $0 \in \tau_F^j(I\!N)$ for every $j \geq 0$. But $\bigcap_{j=0}^{\omega} \tau_F^j(I\!N) = \{0\}$ is not a fixpoint of τ_F since $\tau_2(\{0\}) = \emptyset$, which implies $\tau_F(\{0\}) = \emptyset$. Hence τ_F is not continuous.

On the positive side, syntactical restrictions on F can be identified which guarantee the continuity of τ_F. A well-known example is the condition of "syntactic continuity" which requires that no proposition variable be in the scope of a negation symbol. In particular, τ_F is continuous if F is in positive normal form (defined in section 2.5 below) and contains either only ν- or only μ-operators. Also, non-continuous maps arise only in "pathological" temporal structures whereas every satisfiable νTL formula is satisfiable in some eventually periodic structure which induces a continuous mapping τ_F. This fact has been employed in Banieqbal & Barringer [10] to develop a tableau-based decision procedure for propositional νTL.

Derived temporal operators

The semantic definitions for νTL formulas are "global" in the sense that they yield the set of all states of the structure satisfying the respective formula. Usually, the semantics of temporal logics is defined more "locally" by evaluating formulas in a single state instead over the entire temporal structure (cf. Kröger [29]). However, there is an obvious connection between these two styles of definitions which has already surfaced in the use of the notation $\mathcal{K}_i^{\eta,\xi} \models F$ instead of $i \in \mathcal{K}^{\eta,\xi}(F)$. In fact, the temporal logic operators used in Kröger [29] can be introduced as derived operators in νTL:

$$\begin{aligned} \Box F &\equiv \nu X.F \wedge \circ X \\ \Diamond F &\equiv \mu X.F \vee \circ X \\ F \ \mathsf{atnext}\ G &\equiv \nu X.(G \rightarrow F) \wedge (\neg G \rightarrow \circ X) \end{aligned}$$

$$\begin{aligned} F \;\mathsf{unless}\; G &\equiv \nu X.\neg G \rightarrow (F \wedge \circ X) \\ F \;\mathsf{until}\; G &\equiv \mu X.\neg G \rightarrow (F \wedge \circ X) \end{aligned}$$

(where X is a "new" proposition variable which does not appear in F or G).

The νTL definitions of these operators follow the form of their recursive characterizations given in KRÖGER [29] where "weak" operators (which do not contain an eventuality part) are expressed using the ν-operator whereas "strong" operators such as $\Diamond$ or until are defined as least fixpoints. This connection will become clearer in section 2.6.

Of course, to make the above definitions sensible, we have to prove that these operators in fact behave like their counterparts in KRÖGER [29].

Fact 2.14 *For every temporal structure $\mathcal{K}$ and all valuations η and ξ the following equivalences hold:*

$$\begin{aligned} \mathcal{K}_i^{\eta,\xi} \models \Box F &\iff \mathcal{K}_j^{\eta,\xi} \models F \textit{ for every } j \geq i \\ \mathcal{K}_i^{\eta,\xi} \models \Diamond F &\iff \mathcal{K}_j^{\eta,\xi} \models F \textit{ for some } j \geq i \\ \mathcal{K}_i^{\eta,\xi} \models F \;\mathsf{atnext}\; G &\iff \mathcal{K}_j^{\eta,\xi} \not\models G \textit{ for every } j \geq i \textit{ or } \mathcal{K}_k^{\eta,\xi} \models F \\ & \qquad \textit{for the smallest } k \geq i \textit{ such that } \mathcal{K}_k^{\eta,\xi} \models G \text{ [2]} \\ \mathcal{K}_i^{\eta,\xi} \models F \;\mathsf{unless}\; G &\iff \mathcal{K}_j^{\eta,\xi} \models F \textit{ for every } j \geq i \textit{ or} \\ & \qquad \textit{there is some } k \geq i \textit{ such that } \mathcal{K}_k^{\eta,\xi} \models G \\ & \qquad \textit{and } \mathcal{K}_j^{\eta,\xi} \models F \textit{ for every } i \leq j < k \\ \mathcal{K}_i^{\eta,\xi} \models F \;\mathsf{until}\; G &\iff \mathcal{K}_k^{\eta,\xi} \models G \textit{ for some } k \geq i \textit{ and } \mathcal{K}_j^{\eta,\xi} \models F \\ & \qquad \textit{for every } i \leq j < k \end{aligned}$$

Proof: Let us prove the claim for the case of $\Box F$, the other cases being similar: Assume $i \in \mathcal{K}^{\eta,\xi}(\nu X.F \wedge \circ X)$. Thus, there is some $I \subseteq I\!N$ such that $I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F \wedge \circ X)$ and $i \in I$. Hence, $I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F) \cap \mathcal{K}^{\eta,\xi[I/X]}(\circ X)$, i.e. (since X does not occur in F as a free proposition variable), $(1): I \subseteq \mathcal{K}^{\eta,\xi}(F)$ and $(2): I \subseteq \{k \mid k+1 \in I\}$. From (2) and $i \in I$ one obtains $j \in I$ for every $j \geq i$, and from (1), $\mathcal{K}_j^{\eta,\xi} \models F$ for every $j \in I$ which proves the "only if" direction of the claim. Conversely, assume $\mathcal{K}_j^{\eta,\xi} \models F$ for every $j \geq i$ and let $J := \{j \in I\!N \mid j \geq i\}$. Then, clearly, $J \subseteq \mathcal{K}^{\eta,\xi}(F) = \mathcal{K}^{\eta,\xi[J/X]}(F)$ and $J \subseteq \{k \mid k+1 \in J\}$, hence $J \subseteq \mathcal{K}^{\eta,\xi[J/X]}(F) \cap \mathcal{K}^{\eta,\xi[J/X]}(\circ X) = \mathcal{K}^{\eta,\xi[J/X]}(F \wedge \circ X)$.

[2] Notice that the clauses for the binary operators like atnext given here are slightly different from the definitions of the corresponding operators in KRÖGER [29] in that they are "reflexive", i.e., their truth value in some state also depends on this state. KRÖGER's atnext, e.g., would rather be defined by $\nu X.\circ(G \rightarrow F) \wedge \circ(\neg G \rightarrow X)$.

In particular, $J \subseteq \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F \wedge \circ X)\} = \mathcal{K}^{\eta,\xi}(\Box F)$. Since $i \in J$, it follows that $\mathcal{K}_i^{\eta,\xi} \models \Box F$. ♠

Subsequently, the additional temporal operators defined above will freely be used as derived νTL operators. The operator precedence of $\Box$ and $\Diamond$ is between those of $\circ$ and $\wedge$, the precedence of the binary operators is between $\vee$ and $\rightarrow$.

2.4 Elementary properties of νTL

Let us state some elementary facts about the logic νTL. The first lemma gives an obvious result about the semantics of substitution:

Lemma 2.15 *Let F, G be any formulas and t be a term which does not contain flexible variables. Then*

- $\mathcal{K}^{\eta,\xi}(F[G/X]) = \mathcal{K}^{\eta,\xi[[\![G]\!]/X]}(F)$ *where* $[\![G]\!]$ *denotes* $\mathcal{K}^{\eta,\xi}(G)$.
- $\mathcal{K}^{\eta,\xi}(F[t/y]) = \mathcal{K}^{\eta[[\![t]\!]/y],\xi}(F)$ *where* $[\![t]\!] := \mathcal{K}_0^{\eta}(t)$

Proof: The lemma is proven by a straightforward induction on F. As an example, only the case $F \equiv \nu Y.A$ is given for the first claim: If $Y = X$, $F[G/X] \equiv F$ and there is nothing to be shown. Else, $F[G/X] = \nu Y.A[G/X]$, and by the renaming assumption, there is no free occurrence of Y in G, hence $[\![G]\!] = \mathcal{K}^{\eta,\xi}(G) = \mathcal{K}^{\eta,\xi[I/Y]}(G)$ for any set $I \subseteq I\!N$. Now,

$$\begin{array}{ll} & \mathcal{K}^{\eta,\xi}(\nu Y.A[G/X]) \\ = & \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/Y]}(A[G/X])\} \\ = & \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/Y][[\![G]\!]/X]}(A)\} \quad \text{(by ind. hyp. and the above)} \\ = & \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[[\![G]\!]/X][I/Y]}(A)\} \quad (Y \neq X) \\ = & \mathcal{K}^{\eta,\xi[[\![G]\!]/X]}(\nu Y.A) \end{array}$$ ♠

Our next lemma says that the truth of any formula in some state does not depend on the prefix of the temporal structure up to that state. Stated informally, no formula "depends on the past".

Lemma 2.16 *Let $\mathcal{K} = (\mathcal{S}, \sigma)$ be a temporal structure and let $\mathcal{K}[i..]$ denote the structure $(\mathcal{S}, \sigma[i..])$. Then for all formulas F, all valuations η and ξ and all $j \in I\!N$:*

$$j \in \mathcal{K}[i..]^{\eta,\xi}(F) \iff i + j \in \mathcal{K}^{\eta,\xi}(F)$$

Proof: straightforward induction on F ♠

Lemma 2.17 *Let $\mathcal{K}$ be any temporal structure and η, ξ be valuations of the rigid subject variables and the proposition variables, respectively. Then, for all formulas F and G,*

$$\mathcal{K}^{\eta,\xi}(F) \subseteq \mathcal{K}^{\eta,\xi}(G) \iff \mathcal{K}_i^{\eta,\xi} \models F \to G \textit{ for every } i \in \mathbb{N}.$$

Proof: $\mathcal{K}^{\eta,\xi}(F) \subseteq \mathcal{K}^{\eta,\xi}(G)$
$\iff$ $\mathbb{N} = (\mathbb{N} \setminus \mathcal{K}^{\eta,\xi}(F)) \cup \mathcal{K}^{\eta,\xi}(G) = \mathcal{K}^{\eta,\xi}(F \to G)$
$\iff$ $\mathcal{K}_i^{\eta,\xi} \models F \to G$ for every $i \in \mathbb{N}$ ♠

For the definition of validity of νTL formulas the so-called "initial semantics" of temporal logics (cf. Kröger [30]) has been adopted: A formula F is called true in a temporal structure $\mathcal{K}$ if it holds in the initial state of $\mathcal{K}$. An alternative definition which, e.g., has been employed in Kröger [29] and which has been termed "normal semantics" in Kröger [30], states that F is true in $\mathcal{K}$ if it holds in all states of $\mathcal{K}$, i.e., if $\Box F$ is true in $\mathcal{K}$ according to the definition used here. It is well known that both styles of definition lead to the same set of valid formulas (whereas the corresponding notions of semantic consequence are different). The following lemma establishes this result for νTL.

Lemma 2.18 *Let F be a closed and quantifier closed formula. Then*

$$\models F \iff \models \Box F.$$

Proof:
$\Longrightarrow$: Let F be a valid formula, $\mathcal{K}$, η, ξ be an arbitrary temporal structure and valuations of rigid subject and proposition variables, respectively, and $i \in \mathbb{N}$. Since F is valid, $\mathcal{K}[i..] \models F$, i.e., $0 \in \mathcal{K}[i..]^{\eta,\xi}(F)$ (where $\mathcal{K}[i..]$ is defined as in lemma 2.16). Lemma 2.16 yields $i \in \mathcal{K}^{\eta,\xi}(F)$, and since $i \in \mathbb{N}$ was arbitrary, $\mathcal{K}^{\eta,\xi} \models \Box F$. Since $\mathcal{K}$, η and ξ, too, were arbitrary, $\models \Box F$ follows.
$\Longleftarrow$: trivially from the definition. ♠

The following fact gives some valid formulas which can be used as axioms in a proof system for νTL. Their validity can be established by routine arguments (cf. Kröger [29]).

Fact 2.19

- *Let F be a classical propositional tautology and let F^* result from F by (consistently) replacing the proposition letters in F by νTL formulas. Then F^* is a valid νTL formula.*

- $\models \neg\circ F \leftrightarrow \circ\neg F$
- $\models \circ(F \; op \; G) \leftrightarrow (\circ F \; op \circ G) \quad \textit{where } op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

Lemma 2.20 *Let $\mathcal{F}$ be a set of closed and quantifier closed formulas, and let F, G be closed and quantifier closed formulas. Then*

$$\mathcal{F} \cup \{F\} \models G \quad \Longleftrightarrow \quad \mathcal{F} \models F \rightarrow G$$

Proof:
$\Longrightarrow$: Assume $\mathcal{F} \cup \{F\} \models G$ and let $\mathcal{K}$ be any structure such that $\mathcal{K} \models A$ for all $A \in \mathcal{F}$. If $\mathcal{K}_0 \models F$ then $\mathcal{K}_0 \models G$, hence also $\mathcal{K}_0 \models F \rightarrow G$, since $\mathcal{F} \cup \{F\} \models G$ was assumed. Else, $\mathcal{K}_0 \models \neg F$ (since F is a closed formula), implying $\mathcal{K}_0 \models F \rightarrow G$ as required.

$\Longleftarrow$: Assume $\mathcal{F} \models F \rightarrow G$ and let $\mathcal{K}$ be a temporal structure such that $\mathcal{K} \models A$ for all $A \in \mathcal{F}$ and, moreover, $\mathcal{K} \models F$. By the assumption, $\mathcal{K} \models G$ follows. ♠

The following lemma is a syntactical version of the monotonicity assertion for both fixpoint operators (using the equivalence of lemma 2.17).

Lemma 2.21 $\models F \rightarrow G \quad \Longrightarrow \quad \models \sigma X.F \rightarrow \sigma X.G \quad$ *where $\sigma \equiv \nu$ or $\sigma \equiv \mu$*

Proof: Let $\mathcal{K}$, η and ξ be arbitrary. By the assumption $\models F \rightarrow G$ and lemma 2.17, $\mathcal{K}^{\eta,\xi[I/X]}(F) \subseteq \mathcal{K}^{\eta,\xi[I/X]}(G)$ for every set $I \subseteq \mathbb{N}$.

$$\begin{aligned} \sigma \equiv \nu: \quad \mathcal{K}^{\eta,\xi}(\nu X.F) &= \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F)\} \\ &\subseteq \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(G)\} \quad \text{(see above)} \\ &= \mathcal{K}^{\eta,\xi}(\nu X.G) \end{aligned}$$

$\sigma \equiv \mu$: Assume $i \notin \mathcal{K}^{\eta,\xi}(\mu X.G) = \bigcap\{I \subseteq \mathbb{N} \mid \mathcal{K}^{\eta,\xi[I/X]}(G) \subseteq I\}$. Then there is some $I \subseteq \mathbb{N}$ such that $\mathcal{K}^{\eta,\xi[I/X]}(G) \subseteq I$ and $i \notin I$. Since $\mathcal{K}^{\eta,\xi[I/X]}(F) \subseteq \mathcal{K}^{\eta,\xi[I/X]}(G)$, it follows that $\mathcal{K}^{\eta,\xi[I/X]}(F) \subseteq I$ and $i \notin I$. Hence $i \notin \bigcap\{I \subseteq \mathbb{N} \mid \mathcal{K}^{\eta,\xi[I/X]}(F) \subseteq I\} = \mathcal{K}^{\eta,\xi}(\mu X.F)$.

In both cases, $\mathcal{K}^{\eta,\xi}(\sigma X.F) \subseteq \mathcal{K}^{\eta,\xi}(\sigma X.G)$ holds for arbitrary $\mathcal{K}$, η and ξ. Now, lemma 2.17 yields $\models \sigma X.F \rightarrow \sigma X.G$. ♠

The main result of this section is contained in the following theorem which allows fixpoint formulas to be "unwound" and gives induction rules for such formulas.

Theorem 2.22 (unwinding, fixpoint induction)
Let F and G be formulas, $\mathcal{K}$ be a temporal structure and η and ξ be valuations of the rigid subject variables and the proposition variables, respectively. Then,

(i) $\mathcal{K}^{\eta,\xi}(\sigma X.F) = \mathcal{K}^{\eta,\xi}(F[\sigma X.F/X])$ *for* $\sigma \equiv \nu$ *and* $\sigma \equiv \mu$,

(ii) $\mathcal{K}^{\eta,\xi} \models \Box(G \to F[G/X]) \implies \mathcal{K}^{\eta,\xi} \models \Box(G \to \nu X.F)$
$\mathcal{K}^{\eta,\xi} \models \Box(F[G/X] \to G) \implies \mathcal{K}^{\eta,\xi} \models \Box(\mu X.F \to G)$

Proof:

(i) $[\![\sigma X.F]\!] := \mathcal{K}^{\eta,\xi}(\sigma X.F)$ is a fixpoint of the mapping $2^{\mathbb{N}} \to 2^{\mathbb{N}}, I \mapsto \mathcal{K}^{\eta,\xi[I/X]}(F)$. Hence, $\mathcal{K}^{\eta,\xi}(\sigma X.F) = \mathcal{K}^{\eta,\xi[[\![\sigma X.F]\!]/X]}(F) = \mathcal{K}^{\eta,\xi}(F[\sigma X.F/X])$ (by lemma 2.15).

(ii) Assume $\mathcal{K}^{\eta,\xi} \models \Box(G \to F[G/X])$. Let $[\![G]\!]$ denote $\mathcal{K}^{\eta,\xi}(G)$, then $[\![G]\!] \subseteq \mathcal{K}^{\eta,\xi}(F[G/X]) = \mathcal{K}^{\eta,\xi[[\![G]\!]/X]}(F)$ (using lemmas 2.17 and 2.15). Hence $[\![G]\!] \subseteq \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F)\} = \mathcal{K}^{\eta,\xi}(\nu X.F)$. But this implies $\mathcal{K}^{\eta,\xi} \models \Box(G \to \nu X.F)$. The proof for $\sigma \equiv \mu$ is similar.

♠

2.5 Normal forms for νTL formulas

The syntax of νTL formulas allows almost unrestricted nesting of the basic connectives, the only exceptions being the restriction on quantified subformulas $\forall y F$ and the polarity requirement for proposition variables bound by fixpoint operators. In this section we study some restrictions on the syntax of formulas which do not decrease the expressiveness of νTL. For this purpose, three *normal forms* for νTL formulas will be introduced: positive normal form, non-reflexive normal form and disjunctive normal form (the latter one only for propositional νTL formulas). The requirements for these normal forms are stated in a hierarchical fashion with positive normal form being least restrictive and disjunctive normal form least permissive. Most formulas appearing in the sequel will be (assumed to be) in non-reflexive normal form whereas disjunctive normal form will be used only rarely.

Positive normal form

Positive normal form restricts the scope of the negation symbol to atomic formulas only. Formally, positive normal form is defined as follows:

Definition 2.23 (positive normal form)

- *A* literal *is either an atomic formula* $P(t_1, \ldots, t_n)$ *or* $t_1 = t_2$ *where* P *is an* n*-ary predicate symbol and* t_i *are terms, the negation of such a formula, a proposition variable* $X \in \mathcal{X}$*, or a propositional constant* t *or* f.

- *A* formula in positive normal form *(a PNF formula) is built from literals by finitely many applications of the operators* $\wedge$, $\vee$, $\forall$, $\exists$, $\circ$, ν *and* μ *where* $\forall y F$ *and* $\exists y F$ *are well-formed PNF formulas only if* F *does not contain any free occurrences of proposition variables.*

It is easy to verify that for every νTL formula there is an equivalent PNF formula by observing the following equivalences which, when oriented from left to right, form a rewrite system which actually accomplishes the necessary transformation:

$$\begin{array}{rclcrcl}
\neg\nu X.F & \equiv & \mu X.\neg F[\neg X/X] & \qquad & \neg\mu X.F & \equiv & \nu X.\neg F[\neg X/X] \\
\neg\forall y F & \equiv & \exists y \neg F & & \neg\exists y F & \equiv & \forall y \neg F \\
\neg\circ F & \equiv & \circ\neg F & & \neg\neg F & \equiv & F \\
F \rightarrow G & \equiv & \neg F \vee G & & \neg(F \rightarrow G) & \equiv & F \wedge \neg G \\
\neg(F \wedge G) & \equiv & \neg F \vee \neg G & & \neg(F \vee G) & \equiv & \neg F \wedge \neg G \\
\neg\mathsf{t} & \equiv & \mathsf{f} & & \neg\mathsf{f} & \equiv & \mathsf{t}
\end{array}$$

Observe that the polarity restriction for subformulas $\nu X.F$ or $\mu X.F$ holds trivially for PNF formulas since proposition variables never appear in the scope of a negation symbol. Therefore, the monotonicity lemma 2.13 can be restated in a simpler form. Here, we give an obvious corollary whose formulation will be more useful subsequently:

Lemma 2.24 *Let* F, G, H *be PNF formulas and* X *be any proposition variable. Then, for every temporal structure* $\mathcal{K}$ *and all valuations* η *and* ξ *of the rigid subject and proposition variables, respectively,*

$$[\![G]\!] \subseteq [\![H]\!] \quad \Longrightarrow \quad [\![F[G/X]]\!] \subseteq [\![F[H/X]]\!]$$

(where, for a formula A, $[\![A]\!]$ *again denotes* $\mathcal{K}^{\eta,\xi}(A)$*).*

Proof: Apply lemmas 2.13 and 2.15. ♠

Non-reflexive normal form

The next restriction on the formation of νTL formulas concerns occurrences of proposition variables which are not guarded by some $\circ$ operator as, for example, in $\nu X.p \wedge X$. It will be shown that such occurrences of proposition variables can be eliminated (an observation which has also been made in Banieqbal & Barringer [10]).

Definition 2.25 (non-reflexive normal form) *A PNF formula F is in* non-reflexive normal form *(a NRNF formula) if, for every subformula $\sigma X.G$ of F (where $\sigma \in \{\nu, \mu\}$), every free occurrence of X in G is in the scope of at least one $\circ$ operator.*

Thus, e.g., $\forall y(P(y) \to (\nu X.Q(y) \to \circ(R(y) \wedge X)))$ is a NRNF formula whereas $\forall y(P(y) \to \circ(\nu X.Q(y) \to (R(y) \wedge X)))$ is not. Again, for every PNF formula (hence, for every νTL formula), an equivalent NRNF formula may be obtained by a purely syntactical transformation as is shown by the following lemma:

Lemma 2.26 *For every PNF formula $\sigma X.F$ (where $\sigma \in \{\mu, \nu\}$), there is an equivalent formula $\sigma X.F'$ such that all free occurrences of X in F' appear in the scope of at least one $\circ$ operator. F' can be obtained by replacing every free occurrence of X in F which does not appear in the scope of at least one $\circ$ operator by t if $\sigma \equiv \nu$ and by f otherwise.*

Proof: Let us only give the proof for $\sigma \equiv \nu$, the other case being dual. Let F and F' be as in the assertion of the lemma and let G be a subformula of F. G' results from G as F' from F. Let $\mathcal{K}$, η and ξ be arbitrary and $I \subseteq I\!N$ such that $I = \mathcal{K}^{\eta,\xi[I/X]}(F')$. Then the following assertion is proven:

$$(*) \quad \mathcal{K}^{\eta,\xi[I/X]}(G) \cap I = \mathcal{K}^{\eta,\xi[I/X]}(G') \cap I$$

In particular, choosing $G \equiv F$, $\mathcal{K}^{\eta,\xi[I/X]}(F) \cap I = \mathcal{K}^{\eta,\xi[I/X]}(F') \cap I = I$, i.e., $\mathcal{K}^{\eta,\xi[I/X]}(F') = I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F)$ for every $I \subseteq I\!N$ such that $I = \mathcal{K}^{\eta,\xi[I/X]}(F')$. This, in turn, implies $\mathcal{K}^{\eta,\xi}(\nu X.F') = \bigcup\{I \subseteq I\!N \mid I = \mathcal{K}^{\eta,\xi[I/X]}(F')\} \subseteq \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F)\} = \mathcal{K}^{\eta,\xi}(\nu X.F)$. On the other hand, for every $I \subseteq I\!N$, $\mathcal{K}^{\eta,\xi[I/X]}(X) = I \subseteq I\!N = \mathcal{K}^{\eta,\xi[I/X]}(\mathsf{t})$, and by the monotonicity result of lemma 2.24, it follows that $\mathcal{K}^{\eta,\xi[I/X]}(F) \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F')$ for every $I \subseteq I\!N$ and, consequently, $\mathcal{K}^{\eta,\xi}(\nu X.F) = \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F)\} \subseteq \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(F')\} = \mathcal{K}^{\eta,\xi}(\nu X.F')$ which completes the proof of the lemma.

The assertion $(*)$ is proven by structural induction on G:

- If G is an atomic formula $G \not\equiv X$, or if $G \equiv \circ H$ for some formula H, then $G' \equiv G$, hence the claim trivially holds. The same argument holds for $G \equiv \forall y H$ or $G \equiv \exists y H$ since in those cases H cannot contain any free occurrence of X.

- If $G \equiv X$, $\mathcal{K}^{\eta,\xi[I/X]}(X) \cap I = I = \mathcal{K}^{\eta,\xi[I/X]}(\mathsf{t}) \cap I$.

- Let $G \equiv A \vee B$. Then

$$\begin{array}{cl} & \mathcal{K}^{\eta,\xi[I/X]}(G) \cap I \\ = & (\mathcal{K}^{\eta,\xi[I/X]}(A) \cap I) \cup (\mathcal{K}^{\eta,\xi[I/X]}(B) \cap I) \\ \stackrel{\text{I.H.}}{=} & (\mathcal{K}^{\eta,\xi[I/X]}(A') \cap I) \cup (\mathcal{K}^{\eta,\xi[I/X]}(B') \cap I) \\ = & \mathcal{K}^{\eta,\xi[I/X]}(G') \cap I \end{array}$$

- The proof for $G \equiv A \wedge B$ is similar (substitute $\cap$ for $\cup$).

- Let $G \equiv \nu Y.A$ where $Y \neq X$ (otherwise, again $G' \equiv G$, and there is nothing to prove).

$$\begin{array}{cll} & I \cap \mathcal{K}^{\eta,\xi[I/X]}(G) & \\ = & I \cap \bigcup\{J \subseteq I\!N \mid J = \mathcal{K}^{\eta,\xi[I/X][J/Y]}(A)\} & \\ = & \bigcup\{I \cap J \mid J = \mathcal{K}^{\eta,\xi[I/X][J/Y]}(A)\} & \\ = & \bigcup\{I \cap J \mid I \cap J = I \cap \mathcal{K}^{\eta,\xi[I/X][J/Y]}(A)\} & \text{(by elem. set theory)} \end{array}$$

  Since $X \neq Y$, $\xi[I/X][J/Y] = \xi[J/Y][I/X]$. W.l.o.g., assume that Y does not occur in F or F' as a free variable. Therefore, $I = \mathcal{K}^{\eta,\xi[I/X]}(F') = \mathcal{K}^{\eta,\xi[J/Y][I/X]}(F')$, and the induction hypothesis applies to $\xi[J/Y]$ as well. Continuing in the above derivation we obtain

$$\begin{array}{cll} \stackrel{\text{I.H.}}{=} & \bigcup\{I \cap J \mid I \cap J = I \cap \mathcal{K}^{\eta,\xi[J/Y][I/X]}(A')\} & \\ = & I \cap \bigcup\{J \subseteq I\!N \mid J = \mathcal{K}^{\eta,\xi[I/X][J/Y]}(A')\} & \text{(as above)} \\ = & I \cap \mathcal{K}^{\eta,\xi[I/X]}(\nu Y.A') & \end{array}$$

- The proof for $G \equiv \mu Y.A$ is similar ($\bigcup$ has to be replaced by $\bigcap$).

♠

The preceding lemma not only establishes the existence of some equivalent NRNF formula F' for a given PNF formula F but also gives a simple algorithm for obtaining F'. Thus, e.g.,

$$\begin{array}{lllll} \nu X.A \wedge X & \equiv \nu X.A \wedge \mathsf{t} & \equiv A, & \nu X.A \vee X \equiv \nu X.A \vee \mathsf{t} & \equiv \mathsf{t}, \\ \mu X.A \vee X & \equiv \mu X.A \vee \mathsf{f} & \equiv A, & \mu X.A \wedge X \equiv \mu X.A \wedge \mathsf{f} & \equiv \mathsf{f}. \end{array}$$

Disjunctive normal form

As announced above, the most restrictive normal form to be introduced here is a disjunctive normal form for propositional νTL formulas.

Definition 2.27 (disjunctive normal form) *A propositional NRNF formula F is in* disjunctive normal form *(a DNF formula) if F is of the form $C_1 \vee \ldots \vee C_n$ for some $n \geq 0$ (where the empty disjunction, as usual, represents* f*), and every C_i is of the form $L_i^1 \wedge \ldots \wedge L_i^{n_i}$ for some $n_i > 0$ such that every L_i^j is either*

- *$\circ^k p$ or $\circ^k \neg p$ where p is some proposition letter p or a propositional constant* t *or* f *and $k \geq 0$, or*
- *$\circ^k X$ for some proposition variable $X \in \mathcal{X}$ and some $k \geq 0$, or*
- *$\circ^k \sigma X.B$ where B is a DNF formula, $\sigma \in \{\mu, \nu\}$ and $k \geq 0$.*

The construction of an equivalent DNF formula F' for a given NRNF formula F proceeds similarly as in classical propositional logic by applying the equivalences (oriented from left to right) to every subformula F to which they are applicable:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\circ(A \wedge B) & \equiv & \circ A \wedge \circ B & \circ(A \vee B) & \equiv & \circ A \vee \circ B \\
A \wedge (B \vee C) & \equiv & (A \wedge B) \vee (A \wedge C) & & &
\end{array}$$

It is easily verified that every formula finally obtained by repeated applications of these equivalences is indeed a DNF formula.

Example 2.28 Let us conclude this section about normal forms by giving an example for the successive transformation of a propositional νTL formula to positive, non-reflexive, and disjunctive normal form: Consider the formula $F \equiv \neg\mu X.\circ(p \rightarrow (X \wedge (\nu Y.\neg(q \wedge \neg Y) \wedge (r \rightarrow \circ X))))$. First, an equivalent PNF formula will be derived applying the given equivalences:

$$\begin{array}{rcl}
F & \equiv & \neg\mu X.\circ(p \rightarrow (X \wedge (\nu Y.\neg(q \wedge \neg Y) \wedge (r \rightarrow \circ X)))) \\
 & \equiv & \nu X.\neg\circ(p \rightarrow (\neg X \wedge (\nu Y.\neg(q \wedge \neg Y) \wedge (r \rightarrow \circ\neg X)))) \\
 & \equiv & \nu X.\circ\neg(p \rightarrow (\neg X \wedge (\nu Y.\neg(q \wedge \neg Y) \wedge (r \rightarrow \circ\neg X)))) \\
 & \equiv & \nu X.\circ(p \wedge \neg(\neg X \wedge (\nu Y.\neg(q \wedge \neg Y) \wedge (r \rightarrow \circ\neg X)))) \\
 & \equiv & \nu X.\circ(p \wedge (\neg\neg X \vee \neg(\nu Y.\neg(q \wedge \neg Y) \wedge (r \rightarrow \circ\neg X)))) \\
 & \equiv & \nu X.\circ(p \wedge (X \vee \mu Y.\neg(\neg(q \wedge \neg\neg Y) \wedge (r \rightarrow \circ\neg X)))) \\
 & \equiv & \nu X.\circ(p \wedge (X \vee \mu Y.(\neg\neg(q \wedge \neg\neg Y) \vee \neg(r \rightarrow \circ\neg X))))
\end{array}$$

$$\begin{aligned}
&\equiv \nu X.\circ(p \wedge (X \vee \mu Y.((q \wedge Y) \vee (r \wedge \neg\circ\neg X))))\\
&\equiv \nu X.\circ(p \wedge (X \vee \mu Y.((q \wedge Y) \vee (r \wedge \circ\neg\neg X))))\\
&\equiv \nu X.\circ(p \wedge (X \vee \mu Y.((q \wedge Y) \vee (r \wedge \circ X))))
\end{aligned}$$

Continuing the derivation to obtain an equivalent NRNF formula, we observe that the occurrence of the proposition variable Y in $\mu Y.((q \wedge Y) \vee (r \wedge \circ X))$ is not in the scope of any $\circ$ operator and may be replaced by f to yield an equivalent formula by lemma 2.26. After this substitution, Y does not occur anymore in the resulting formula. Applying the "unwinding" transformation of theorem 2.22(i) and propositional simplification, we arrive at the formula

$$F' \equiv \nu X.\circ(p \wedge (X \vee (r \wedge \circ X)))$$

which is equivalent to the original formula F and in non-reflexive normal form. Continuing from this formula to produce an equivalent DNF formula, we obtain

$$\begin{aligned}
F' &\equiv \nu X.(\circ p \wedge \circ(X \vee (r \wedge \circ X)))\\
&\equiv \nu X.(\circ p \wedge (\circ X \vee \circ(r \wedge \circ X)))\\
&\equiv \nu X.(\circ p \wedge (\circ X \vee (\circ r \wedge \circ\circ X)))\\
&\equiv \nu X.(\circ p \wedge \circ X) \vee (\circ p \wedge \circ r \wedge \circ\circ X))).
\end{aligned}$$

2.6 Characterizing safety properties in νTL

This section provides some links between the general classification of properties studied in section 2.1 and the syntactical form of νTL formulas expressing such properties. In particular, a syntactical characterization of (formulas expressing) safety properties will be given. Also, for a certain class of formulas, a formula characterizing the safety closure of (the property expressed by) a given formula can be constructed by a simple syntactical transformation. Let us first establish appropriate notions linking properties and formulas:

Let $\mathcal{K} = (\mathcal{S}, \sigma)$ be a temporal structure for a given language $\mathcal{L}$. Assume, for the remainder of this section, a fixed first-order structure $\mathcal{S}$ to be given. The sequence $\sigma = (\sigma_0, \sigma_1, \ldots)$ has been defined as an ω-sequence of states where each state is a mapping assigning a value to every flexible variable and a subset of $|\mathcal{S}|^n$ to every n-ary predicate symbol of $\mathcal{L}$. The set of all such states will be denoted by Σ, and the temporal structure $\mathcal{K}$ will be identified with its state sequence $\sigma \in \Sigma^\omega$. In particular, if F is a νTL formula, $\sigma^{\eta,\xi}(F)$ will frequently stand for $\mathcal{K}^{\eta,\xi}(F)$. We will moreover apply the notations for prefixes, suffixes and products to temporal structures σ. Call a property P *defined by the formula* F if $P = Mod(F)$ holds in Σ^ω.

Before studying the syntactical form of νTL formulas expressing safety properties, let us state a technical lemma concerning the $\circ$-*fragment* of νTL which is defined as the set of formulas built from atomic formulas $P(t_1, \ldots, t_n)$ or $t_1 = t_2$ by applying propositional connectives like $\neg$ and $\wedge$, first-order quantifiers and the temporal operator $\circ$ only (i.e., νTL-formulas which contain neither proposition variables nor the operators ν or μ). In particular, every such formula F is closed, hence the reference to the valuation ξ of the propositional variables in the definition of the semantics $\sigma^{\eta,\xi}(F)$ of F may be dropped. The lemma states that the truth of such formulas in some state depends only on a finite (future) part of the temporal structure:

Lemma 2.29 *Let F be a formula of the $\circ$-fragment of νTL. Then there is some $k \in I\!N$ such that for every temporal structure σ, all valuations η of the rigid subject variables and every $i \in I\!N$ the following assertion holds:*

$$i \in \sigma^\eta(F) \quad \Longleftrightarrow \quad \forall \tau \in \Sigma^\omega \; i \in (\sigma[..i+k]\tau)^\eta(F)$$

Proof: Only the implication "$\Longrightarrow$" has to be proved. Let k be the number $\mathrm{nex}(F)$ of $\circ$-operators appearing in F. The assertion is proved by induction on the structure of the formula F.

- $F \equiv P(t_1, \ldots, t_n)$: Then $\mathrm{nex}(F) = 0$. Assume $i \in \sigma^\eta(P(t_1, \ldots, t_n))$, i.e., $(\sigma_i^\eta(t_1), \ldots, \sigma_i^\eta(t_n)) \in \sigma_i(P)$. By the definition of term evaluation in temporal structures, $\sigma_i^\eta(t)$ for any term t only depends on the valuation σ_i, hence $\sigma_i^\eta(t) = (\sigma[..i]\tau)_i^\eta(t)$ for any sequence $\tau \in \Sigma^\omega$. Thus, $((\sigma[..i]\tau)_i^\eta(t_1), \ldots, (\sigma[..i]\tau)_i^\eta(t_n)) \in \sigma_i(P) = (\sigma[..i]\tau)_i(P)$ for every $\tau \in \Sigma^\omega$, i.e., $i \in (\sigma[..i]\tau)^\eta(P(t_1, \ldots, t_n))$.
- $F \equiv t_1 = t_2$: similar
- $F \equiv \neg G$: Then $k = \mathrm{nex}(F) = \mathrm{nex}(G)$. Assume $i \in \sigma^\eta(\neg G)$, i.e., $i \notin \sigma^\eta(G)$. If, for some $\tau \in \Sigma^\omega$, $i \in (\sigma[..i+k]\tau)^\eta(G)$, then the induction hypothesis (applied to $\sigma[..i+k]\tau$ instead of σ) would yield that for all $\rho \in \Sigma^\omega$, $i \in ((\sigma[..i+k]\tau)[..i+k]\rho)^\eta(G) = (\sigma[..i+k]\rho)^\eta(G)$, hence, in particular, choosing $\rho = \sigma[i+k+1..]$, $i \in \sigma^\eta(G)$ contradicting the assumption.
- $F \equiv G \wedge H$: Then $\mathrm{nex}(G) \leq \mathrm{nex}(F)$ and $\mathrm{nex}(H) \leq \mathrm{nex}(F)$. Apply the induction hypothesis, noting that if the assertion holds for some $k \in I\!N$ then it also holds for all $m \geq k$.

- $F \equiv \forall x G$: similar
- $F \equiv \circ G$: Then $k = \text{nex}(F) = \text{nex}(G) + 1$. Assume $i \in \sigma^\eta(\circ G)$, i.e., $i+1 \in \sigma^\eta(G)$. By the induction hypothesis, for every $\tau \in \Sigma^\omega$, $i+1 \in (\sigma[..i+1+\text{nex}(G)]\tau)^\eta(G)$, i.e., $i \in (\sigma[..i+k]\tau)^\eta(\circ G)$ which proves the assertion.

♠

Every closed and quantifier closed νTL formula F characterizes a property $Mod(F) \subseteq \Sigma^\omega$, viz. the set of its models. Therefore it is sensible to apply the notions studied in section 2.1 and, in particular, classify (closed and quantifier closed) νTL formulas according to whether the properties expressed by them are safety or liveness properties. For propositional νTL, it has been noticed frequently (e.g., see Larsen [33] or Bradfield & Stirling [13]) — albeit without proving this observation explicitly — that safety properties are expressed by formulas (in positive form) using maximal fixpoint operators whereas minimal fixpoints are indispensable for the formulation of liveness properties. And, in fact, a short inspection of the list of basic operators introduced up to now supports this observation: Formulas like $\Box p$, p **atnext** q, p **unless** q or $\forall y \Box (y = z \rightarrow \circ(y = z + 2 \textbf{ unless } z = 0))$ which do not contain any subformula $\mu X.F$ all express safety properties, whereas $\Diamond q$, $\Box\Diamond z = 0$ or $\forall y (P(y) \textbf{ until } Q(y))$ express either pure liveness properties or properties possessing a nontrivial liveness extension. The following theorem gives a (sufficient) syntactical criterion for formulas expressing safety properties:

Theorem 2.30 *Let F be a closed and quantifier closed PNF formula which does not contain subformulas of the form $\mu X.G$ and where no subformula $\nu X.G$ appears in the scope of an $\exists$ quantifier. Then $Mod(F)$ is a safety property.*

Proof: Let F be any formula of the required form. We have to show that for every $\sigma \in \Sigma^\omega$, if for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $\sigma[..i]\tau \models F$, then $\sigma \models F$. We will show the following more general assertion: Let F be a (not necessarily closed or quantifier closed) PNF formula without any subformula $\mu X.G$ where no subformula $\nu X.G$ occurs in the scope of an existential quantifier. Then for every set $I \subseteq \mathbb{N}$ of natural numbers, every temporal structure $\sigma \in \Sigma^\omega$ and all valuations η and ξ of the rigid subject and the proposition variables, respectively:

$$(*) \quad (\forall i \geq 0 \exists \tau \in \Sigma^\omega : I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(F)) \implies I \subseteq \sigma^{\eta,\xi}(F)$$

The claim of the theorem follows from $(*)$ by choosing $I = \{0\}$ and the assumption that F is closed and quantifier closed. The assertion $(*)$ is proven by structural induction on F. The antecedent of the implication $(*)$ will be assumed.

- $F \equiv (\neg)P$ where P is an atomic formula: Let $i \in I$. By the assumption, there is some $\tau \in \Sigma^\omega$ such that $i \in (\sigma[..i]\tau)^{\eta,\xi}(F)$. In this case, $i \in \sigma^{\eta,\xi}(F)$ follows by similar reasoning as in the proof of lemma 2.29. Since $i \in I$ was chosen arbitrary, it follows that $I \subseteq \sigma^{\eta,\xi}(F)$.

- $F \equiv X$ for some $X \in \mathcal{X}$: By the assumption there is some $\tau \in \Sigma^\omega$ such that $I \subseteq (\sigma[..0]\tau)^{\eta,\xi}(X) = \xi(X) = \sigma^{\eta,\xi}(X)$.

- $F \equiv G \wedge H$: Let $I \subseteq \mathbb{N}$ such that for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(G \wedge H) = (\sigma[..i]\tau)^{\eta,\xi}(G) \cap (\sigma[..i]\tau)^{\eta,\xi}(H)$, i.e., for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(G)$ and $I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(H)$. By the induction hypothesis, $I \subseteq \sigma^{\eta,\xi}(G)$ and $I \subseteq \sigma^{\eta,\xi}(H)$, hence $I \subseteq \sigma^{\eta,\xi}(G \wedge H)$.

- $F \equiv G \vee H$: Assume that, for some $I \subseteq \mathbb{N}$, $I \not\subseteq \sigma^{\eta,\xi}(G \vee H)$. Then $I \not\subseteq \sigma^{\eta,\xi}(G)$ and $I \not\subseteq \sigma^{\eta,\xi}(H)$. By the induction hypothesis there are $j, k \geq 0$ such that for all $\tau \in \Sigma^\omega$, $I \not\subseteq (\sigma[..j]\tau)^{\eta,\xi}(G)$ and $I \not\subseteq (\sigma[..j]\tau)^{\eta,\xi}(H)$. Taking $l = \max(j,k)$, this implies that there is some $l \geq 0$ such that for all $\tau \in \Sigma^\omega$, $I \not\subseteq (\sigma[..l]\tau)^{\eta,\xi}(G \vee H)$.

- $F \equiv \forall y G$:
$$\begin{aligned} & \forall i \geq 0 \exists \tau \in \Sigma^\omega : I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(\forall y G) \\ \Longrightarrow\ & \forall i \geq 0 \exists \tau \in \Sigma^\omega : I \subseteq \textstyle\bigcap_{d \in \mathcal{S}} (\sigma[..i]\tau)^{\eta[d/y],\xi}(G) \\ \Longrightarrow\ & \forall d \in \mathcal{S} \forall i \geq 0 \exists \tau \in \Sigma^\omega : I \subseteq (\sigma[..i]\tau)^{\eta[d/y],\xi}(G) \\ \stackrel{\text{I.H.}}{\Longrightarrow}\ & \forall d \in \mathcal{S} : I \subseteq \sigma^{\eta[d/y],\xi}(G) \\ \Longrightarrow\ & I \subseteq \textstyle\bigcap_{d \in \mathcal{S}} \sigma^{\eta[d/y],\xi}(G) = \sigma^{\eta,\xi}(\forall y G) \end{aligned}$$

- $F \equiv \exists y G$: Then G (hence, also F) is a formula of the $\circ$-fragment of νTL (since G can neither contain free proposition variables nor subformulas $\mu X.H$ or $\nu X.H$ by the syntactic restrictions assumed in the assertion). By lemma 2.29 there is some $k \in \mathbb{N}$ such that for all σ, η and $i \in \mathbb{N}$, $i \in (\sigma[..i+k]\tau)^\eta(F)$ for every $\tau \in \Sigma^\omega$, and in particular $i \in \sigma^\eta(F)$ if there is some $\tau \in \Sigma^\omega$ such that $i \in (\sigma[..i+k]\tau)^\eta(F)$. Assume the antecedent of $(*)$ and let $i \in I$. Then there is some $\tau \in \Sigma^\omega$ such that $i \in (\sigma[..i+k]\tau)^\eta(F)$, hence $i \in \sigma^\eta(F)$ by lemma 2.29. Since $i \in I$ was arbitrary, it follows that $I \subseteq \sigma^\eta(F)$.

- $F \equiv \circ G$: Assume $I \subseteq \mathbb{N}$ such that for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(\circ G) = \{j \mid j+1 \in (\sigma[..i]\tau)^{\eta,\xi}(G)\}$. Let I^{+1} denote the set $\{i+1 \mid i \in I\}$, then $I^{+1} \subseteq (\sigma[..i]\tau)^{\eta,\xi}(G)$, hence, applying the induction hypothesis for I^{+1}, it follows that $I^{+1} \subseteq \sigma^{\eta,\xi}(G)$, i.e., $I \subseteq \sigma^{\eta,\xi}(\circ G)$.

- $F \equiv \nu X.G$: Assume $I \subseteq I\!N$ such that for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $I \subseteq (\sigma[..i]\tau)^{\eta,\xi}(\nu X.G) = \bigcup\{J \subseteq I\!N \mid J \subseteq (\sigma[..i]\tau)^{\eta,\xi[J/X]}(G)\}$. Then for every $k \in I$ there is some set $J_k \subseteq I\!N$ such that $k \in J_k$ and $J_k \subseteq (\sigma[..i]\tau)^{\eta,\xi[J_k/X]}(G)$, hence by the monotonicity result of lemma 2.13, $J_k \subseteq (\sigma[..i]\tau)^{\eta,\xi[(\bigcup_{k\in I} J_k)/X]}(G)$ for every $k \in I$, and consequently $I \subseteq \bigcup_{k\in I} J_k \subseteq (\sigma[..i]\tau)^{\eta,\xi[(\bigcup_{k\in I} J_k)/X]}(G)$. Denoting $\bigcup_{k\in I} J_k$ by M_i, this means that for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ and some $M_i \subseteq I\!N$ such that $I \subseteq M_i$ and $M_i \subseteq (\sigma[..i]\tau)^{\eta,\xi[M_i/X]}(G)$. One may even choose $M_i = M_{i+1}$ for every $i \geq 0$: Assuming that there is some $\tau \in \Sigma^\omega$ such that $I \subseteq M_{i+1}$ and $M_{i+1} \subseteq (\sigma[..i+1]\tau)^{\eta,\xi[M_{i+1}/X]}(G)$ it follows that there is $\tau' \in \Sigma^\omega$ such that $I \subseteq M_{i+1}$ and $M_{i+1} \subseteq (\sigma[..i]\tau')^{\eta,\xi[M_{i+1}/X]}(G)$ (choose $\tau' := \sigma[i+1]\tau$). But then, there is some $M \subseteq I\!N$ such that $I \subseteq M$ and for every $i \geq 0$ there is some $\tau \in \Sigma^\omega$ such that $M \subseteq (\sigma[..i]\tau)^{\eta,\xi[M/X]}(G)$. Applying the induction hypothesis for M instead of I and $\xi[M/X]$ instead of ξ yields $M \subseteq \sigma^{\eta,\xi[M/X]}(G)$, hence $I \subseteq M \subseteq \bigcup\{J \subseteq I\!N \mid J \subseteq \sigma^{\eta,\xi[J/X]}(G)\} = \sigma^{\eta,\xi}(\nu X.G)$.

♠

In particular, every closed propositional PNF formula which is built using the ν operator as only fixpoint operator expresses a safety property. To see that the proviso concerning existential quantifiers really is necessary, observe the following example: Let z be a flexible variable ranging over natural numbers and let $\leq$ have its usual (state-independent) interpretation on $I\!N$. Whereas $\Box\exists n\ z \leq n$ clearly is a safety property ($\exists n\ z \leq n$ being a valid formula in this interpretation), $\exists n\ \Box z \leq n$ is not: Let σ be the state sequence where $\sigma_i(z) = i$ for every $i \geq 0$. Then, $\sigma \not\models \exists n\ \Box z \leq n$, but every finite prefix $\sigma[..i]$ can clearly be extended by some τ to yield a model of this formula. This observation is consistent with the result of lemma 2.2 that, in general, only finite unions of safety properties again define a safety property.

Let us in the remainder of this section consider the question if the syntactic condition of theorem 2.30 is — for propositional νTL — not only sufficient but also necessary for a formula to express a safety property. Now, one has to be a bit careful in the formulation of the converse statement for two rather trivial reasons: First, there are properties which are not expressed by any νTL-formula, so the question is only reasonable for the restricted class of *νTL-expressible* properties. (For example, no νTL formula expresses the safety property that p holds at every state $\sigma[i]$ where i is a prime number). And on the other hand, the formulas might contain "inessential" μ-operators such as $\mu X.\circ X$ which is equivalent to f (a safety property). Therefore, a reasonable statement of the converse assertion of theorem 2.30, restricted to propositional νTL, is the following: If, for some formula F, *Mod*(F) is a safety property, does there exist an equivalent PNF formula F' which does not contain any subformulas $\mu X.G$?

Some more notation will be convenient in the following:

Definition 2.31 ($\circ$-degree) *Let F be some propositional PNF formula and A be an (occurrence of a) subformula of F. The* $\circ$-degree *of A in F,* $\circ$-deg$(A, F) \in \mathbb{N}$*, is inductively defined as follows:*

- $\circ$-deg$(F, F) = 0$.
- *If $F \in \{\neg F', F' \wedge G, G \wedge F', F' \vee G, G \vee F', \nu X.F', \mu X.F'\}$ (for some formula G and proposition variable $X \in \mathcal{X}$) where the occurrence of A in F is in F', then* $\circ$-deg$(A, F) = \circ$-deg(A, F').
- $\circ$-deg$(A, \circ F') = \circ$-deg$(A, F') + 1$.

We will introduce an even more restricted normal form for propositional νTL formulas which, unlike the normal forms of section 2.5, is not defined as a purely syntactical restriction: Let $F(\sigma X.G)$ be some NRNF formula such that the occurrence of the subformula $\sigma X.G$ (where $\sigma \in \{\mu, \nu\}$) of F is not in the scope of a $\circ$-operator, i.e., $\circ$-deg$(\sigma X.G, F) = 0$. Then by unwinding (theorem 2.22), F is equivalent to $F(G[\sigma X.G/X])$. Since F (and therefore G) were assumed to be NRNF formulas, the $\circ$-degree of the occurrences of $\sigma X.G$ in $G[\sigma X.G/X]$ will now be at least 1. Proceeding inductively, for every $n \geq 0$ there is some formula $F^{(n)}$ equivalent to F such that the $\circ$-degree of all occurrences of subformulas of the form $\sigma X.G$ in $F^{(n)}$ is at least n. W.l.o.g., we may further assume that $F^{(n)}$ is a DNF formula $F^{(n)} \equiv \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n_i} L_{ij}$. Call $F^{(n)}$ the *n-fold unfolding* of F. Then the formula $G^{(n)}$ which results from $F^{(n)}$ by deleting all L_{ij} whose $\circ$-degree in $F^{(n)}$ is less than n and eliminating empty conjunctions is called the *reduced n-fold unfolding* of F. The original formula F is called a *reduced formula* if either $F \equiv \mathsf{f}$, or for every $n \geq 0$, every disjunct $\bigwedge_{j=1}^{k_i} L_{ij}$ of the reduced n-fold unfolding $G^{(n)}$ of F is satisfiable.

An example for the construction of the formulas $F^{(n)}$ and $G^{(n)}$ will be given below. Intuitively, the reduced n-fold unfolding $G^{(n)}$ of a formula F represents the requirements on the suffix $\sigma[n..]$ of a temporal structure σ for being a model of F. These requirements are stated in the form of a disjunction, and for reduced formulas, every disjunct has at least one model — hence the satisfiability of the corresponding disjuncts in the n-fold unfolding $F^{(n)}$ of F only depends on the finite prefix $\sigma[..n-1]$ of σ. This will be the crucial idea for the proof of lemma 2.35.

Lemma 2.32 *For every propositional νTL formula F there is an equivalent reduced formula F'.*

Proof: Suppose, to the contrary, that for some formula F there is no equivalent reduced formula F'. In particular, F is not reduced, i.e., for some smallest $n \geq 0$, the reduced n-fold unfolding $G^{(n)}$ of F contains an unsatisfiable disjunct $D_i' \equiv \bigwedge_{j=1}^{k_i} L_{ij}$. For any formula A, denote by $|A|_n$ the number of disjuncts of the n-fold unfolding of A. Assume F to be chosen such that for every formula A such that $|A|_n < |F|_n$ there is an equivalent reduced formula A'. Now, for every unsatisfiable disjunct $D_i' \equiv \bigwedge_{j=1}^{k_i} L_{ij}$ of $G^{(n)}$, the n-fold unfolding $F^{(n)}$ of F contains a corresponding disjunct $D_i \equiv \bigwedge_{j=1}^{n_i} L_{ij}$ from which D_i' was obtained by deleting those L_{ij} whose $\circ$-degree in $F^{(n)}$ is less than n. Clearly, all those disjuncts D_i are also unsatisfiable. Therefore, the formula F'' obtained from $F^{(n)}$ by deleting these D_i is equivalent to $F^{(n)}$, and therefore also to the original formula F. Since the $\circ$-degree of all occurrences of subformulas $\sigma X.G$ appearing in $F^{(n)}$ is at least n (by definition of $F^{(n)}$), the same holds for F''. Hence, F'' is its own n-fold unfolding and contains fewer disjuncts than $F^{(n)}$, i.e., $|F''|_n < |F|_n$. By the choice of F it follows that there is some reduced formula F' which is equivalent to F'', hence also to F, contradicting the original assumption. ♠

Example 2.33 Consider the formula $F \equiv \Diamond p \wedge \circ\circ\circ\Box\neg p$. Then the 3-fold unfolding of F is $F^{(3)} \equiv (p \wedge \circ\circ\circ\Box\neg p) \vee (\circ p \wedge \circ\circ\circ\Box\neg p) \vee (\circ\circ p \wedge \circ\circ\circ\Box\neg p) \vee (\circ\circ\circ\Diamond p \wedge \circ\circ\circ\Box\neg p)$, and the reduced 3-fold unfolding of F is $G^{(3)} \equiv \circ\circ\circ\Box\neg p \vee \circ\circ\circ\Box\neg p \vee \circ\circ\circ\Box\neg p \vee (\circ\circ\circ\Diamond p \wedge \circ\circ\circ\Box\neg p)$. $G^{(3)}$ contains the unsatisfiable disjunct $\circ\circ\circ\Diamond p \wedge \circ\circ\circ\Box\neg p$. Hence F is not reduced. A reduced formula which is equivalent to F is $F' \equiv (p \vee \circ p \vee \circ\circ p) \wedge \circ\circ\circ\Box\neg p$. ♠

Remark: A more careful analysis would reveal that there is an upper bound for the number of distinct formulas $G^{(n)}$ which have to be tested: Each disjunct $\bigwedge_{j=1}^{k_i} L_{ij}$ appearing in $G^{(n)}$ is satisfiable if and only if $\bigwedge_{j=1}^{k_i} \bar{L}_{ij}$ is satisfiable where $\bar{L}_{ij}$ is obtained from L_{ij} by deleting the n outermost $\circ$-operators. Denote by $\bar{G}^{(n)}$ the formula resulting from $G^{(n)}$ by this procedure. Evidently, instead of testing the formulas $G^{(n)}$ one can equivalently test $\bar{G}^{(n)}$ for satisfiability of all its disjuncts. Now, all the formulas $\bar{L}_{ij}$ are contained in the Fischer-Ladner closure $Cl(F)$ of F (defined as the set of subformulas of F with the additional clause $Cl(X) = \{X\} \cup Cl(\sigma X.F_X)$ where $\sigma X.F_X$ is the (w.l.o.g. unique) subformula of F where the proposition variable X appears bound, or $Cl(X) = \{X\}$ if X is a free variable of F). In particular, when every formula $\bar{G}^{(n)} \equiv \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} \bar{L}_{ij}$ is identified with the set $\{\{\bar{L}_{11}, \ldots, \bar{L}_{1k_1}\}, \ldots, \{\bar{L}_{m1}, \ldots, \bar{L}_{mk_m}\}\}$, there result only finitely many different $\bar{G}^{(n)}$. This observation is at the base of a tableau decision procedure testing νTL formulas for satisfiability which will not be pursued here (see, e.g., Banieqbal & Barringer [10]). Let us note, however, that from such an analysis it follows that, first, it is decidable whether some propositional νTL formula F is reduced and, moreover, that an effective procedure may be given for constructing an equivalent reduced formula for any propositional νTL formula F.

Let F be a propositional PNF formula and denote by F^ν the formula resulting from F by replacing each μ-operator in F by ν. As may be expected, every model of F also is a model of F^ν:

Lemma 2.34 *Let F be a closed propositional PNF formula. Then*

$$Mod(F) \subseteq Mod(F^\nu).$$

Proof: For the proof, the assumption that F is a closed formula has to be dropped. Therefore we show that for any propositional PNF formula F, all temporal structures σ and all valuations ξ of the proposition variables, $\sigma^\xi(F) \subseteq \sigma^\xi(F^\nu)$: The proof again runs by structural induction:

- $F \equiv L$ where L is a literal:
  Then the assertion holds trivially since $F \equiv F^\nu$.
- $F \equiv G \wedge H$:
  $\sigma^\xi(G \wedge H) = \sigma^\xi(G) \cap \sigma^\xi(H) \stackrel{\text{I.H.}}{\subseteq} \sigma^\xi(G^\nu) \cap \sigma^\xi(H^\nu)$
  $= \sigma^\xi(G^\nu \wedge H^\nu) = \sigma^\xi((G \wedge H)^\nu)$
- $F \equiv G \vee H$: similar
- $F \equiv \circ G$:
  $\sigma^\xi(\circ G) = \{i \mid i+1 \in \sigma^\xi(G)\} \stackrel{\text{I.H.}}{\subseteq} \{i \mid i+1 \in \sigma^\xi(G^\nu)\}$
  $= \sigma^\xi(\circ G^\nu) = \sigma^\xi((\circ G)^\nu)$
- $F \equiv \nu X.G$:
  $\sigma^\xi(\nu X.G) = \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \sigma^{\xi[I/X]}(G)\}$
  $\stackrel{\text{I.H.}}{\subseteq} \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \sigma^{\xi[I/X]}(G^\nu)\} = \sigma^\xi(\nu X.G^\nu) = \sigma^\xi((\nu X.G)^\nu)$
- $F \equiv \mu X.G$:
  $\sigma^\xi(\mu X.G) = \bigcap\{I \subseteq \mathbb{N} \mid I = \sigma^{\xi[I/X]}(G)\}$
  $\subseteq \bigcup\{I \subseteq \mathbb{N} \mid I = \sigma^{\xi[I/X]}(G)\} = \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \sigma^{\xi[I/X]}(G)\}$
  $\stackrel{\text{I.H.}}{\subseteq} \bigcup\{I \subseteq \mathbb{N} \mid I \subseteq \sigma^{\xi[I/X]}(G^\nu)\} = \sigma^\xi(\nu X.G^\nu) = \sigma^\xi((\mu X.G)^\nu)$ ♠

For reduced formulas, moreover, the following assertion can be established:

Lemma 2.35 *Let F be a closed and reduced propositional NRNF formula and $\sigma = (\sigma_0, \ldots, \sigma_n) \in \Sigma^+$ be a finite sequence of states such that for every $\tau \in \Sigma^\omega$, $\sigma\tau \not\models F$. Then $\sigma\tau \not\models F^\nu$ for every $\tau \in \Sigma^\omega$.*

Proof: Consider $F^{(n+1)}$, the $(n+1)$-fold unfolding of F (which by construction is also a closed formula). Since F and $F^{(n+1)}$ are equivalent, it follows from the assumption that for every $\tau \in \Sigma^\omega$, $\sigma\tau \not\models F^{(n+1)}$. Specifically, if $F^{(n+1)} \equiv \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} L_{ij}$, then for every $\tau \in \Sigma^\omega$ and every $1 \leq i \leq m$, $\sigma\tau \not\models \bigwedge_{j=1}^{k_i} L_{ij}$, i.e., $\sigma\tau \models \bigvee_{j=1}^{k_i} \neg L_{ij}$. There are two possible cases:

<u>Case 1:</u> There is some $\tau \in \Sigma^\omega$ such that for every $1 \leq i \leq m$ there is some $1 \leq j \leq k_i$ such that $\sigma\tau \not\models L_{ij}$ and $\circ$-deg$(L_{ij}, F^{(n+1)}) \leq n$. Then all those L_{ij} have the form $\circ^k(\neg)p$ for some proposition letter p (since all subformulas $\nu X.G$ or $\mu X.G$ have a $\circ$-degree of at least $n+1$ by the construction of $F^{(n+1)}$ and proposition variables do not appear outside the scope of formulas $\sigma X.G$ since F, hence also $F^{(n+1)}$, is a closed formula). Now consider the $(n+1)$-fold unfolding $(F^\nu)^{(n+1)}$ of F^ν: By the definition of F^ν and the unfolding of a formula, obviously $(F^\nu)^{(n+1)} = (F^{(n+1)})^\nu = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} L_{ij}^\nu$. Since $(\circ^k(\neg)p)^\nu = \circ^k(\neg)p$ for every proposition letter p, it follows that $\sigma\tau \not\models L_{ij}^\nu$ for every L_{ij} mentioned above. Thus, for every $\tau \in \Sigma^\omega$, $\sigma\tau \not\models (F^\nu)^{(n+1)}$ and, since F^ν and $(F^\nu)^{(n+1)}$ are again equivalent, $\sigma\tau \not\models F^\nu$ as asserted.

<u>Case 2</u>: For every $\tau \in \Sigma^\omega$ there is some $1 \leq i \leq m$ such that $\sigma\tau \models L_{ij}$ for all L_{ij} whose $\circ$-degree in $F^{(n+1)}$ is at most n. Let, w.l.o.g., for $1 \leq i \leq m$, $\{L_{i1}, \ldots, L_{il_i}\}$ be the set of L_{ij} whose $\circ$-degree in $F^{(n+1)}$ is at most n, and $\{L_{il_i+1}, \ldots, L_{ik_i}\}$ the set of remaining L_{ij}. Choose any τ and i such that $\sigma\tau \models \bigwedge_{j=1}^{l_i} L_{ij}$. It has been argued above that for $1 \leq j \leq l_i$, the L_{ij} are of the form $\circ^k p$ for some $k \leq n$ and some proposition letter p, hence $\bigwedge_{j=1}^{l_i} L_{ij}$ is a formula of the propositional $\circ$-fragment of νTL. By the proof of lemma 2.29, it follows that in this case $\sigma\tau \models \bigwedge_{j=1}^{l_i} L_{ij}$ for every $\tau \in \Sigma^\omega$. Hence, by the assumption that $\sigma\tau \not\models \bigwedge_{j=1}^{k_i} L_{ij}$ for every $\tau \in \Sigma^\omega$, it follows that $\sigma\tau \not\models \bigwedge_{j=l_i+1}^{k_i} L_{ij}$ for every $\tau \in \Sigma^\omega$ (in particular, there is at least one such L_{ij}). Now, all those L_{ij} have a $\circ$-degree of at least $n+1$, i.e., their truth in $\sigma\tau$ does not depend on σ. But then $\rho \not\models \bigwedge_{j=l_i+1}^{k_i} L_{ij}$ for all $\rho \in \Sigma^\omega$. Observing that this formula is one of the disjuncts appearing in the reduced $(n+1)$-fold unfolding $G^{(n+1)}$ of F, it follows that $G^{(n+1)}$ contains an unsatisfiable disjunct contradicting the assumption that F was a reduced formula. Hence this case is in fact impossible. ♠

Combining the preceding lemmas, the following theorem follows which characterizes the safety closure of reduced propositional νTL formulas:

Theorem 2.36 *Let F be a closed and reduced propositional νTL formula. Then*

$$safe(Mod(F)) = Mod(F^\nu).$$

Proof: By theorem 2.30, F^ν expresses a safety property, and by lemma 2.34, $Mod(F) \subseteq Mod(F^\nu)$. It remains to be shown that $Mod(F^\nu)$ is the smallest safety property containing $Mod(F)$. Assume P were a safety property (which need not necessarily be expressible as $P = Mod(G)$ for some νTL formula G) such that $Mod(F) \subseteq P$ and $Mod(F^\nu) \nsubseteq P$. Then there is some temporal structure $\sigma \in Mod(F^\nu)$ such that $\sigma \notin P$. Since P is a safety property, there is some $n \in I\!N$ such that $\sigma[..n]\tau \notin P$ for every $\tau \in \Sigma^\omega$. By $Mod(F) \subseteq P$ it follows that $\sigma[..n]\tau \notin Mod(F)$ for every $\tau \in \Sigma^\omega$, hence $\sigma[..n]\tau \notin Mod(F^\nu)$ for every $\tau \in \Sigma^\omega$ by lemma 2.35, contradicting the assumption $\sigma \in Mod(F^\nu)$. ♠

As an immediate corollary, the above question concerning the converse of theorem 2.30 can be resolved in a positive way (at least for propositional νTL):

Corollary 2.37 *For every closed propositional νTL formula F such that $Mod(F)$ is a safety property, there is an equivalent formula G such that G does not contain any subformula $\mu X.H$.*

Proof: W.l.o.g., F may be assumed to be a NRNF formula. Then by lemma 2.32 there is an equivalent reduced formula F'. In particular, $Mod(F')$ also is a safety property and by lemma 2.10 and theorem 2.36, $Mod(F') = safe(Mod(F')) = Mod((F')^\nu)$, i.e., F' (hence, F) is equivalent to $(F')^\nu$. Taking $G \equiv (F')^\nu$, the assertion follows. ♠

Moreover, the liveness part of reduced propositional νTL formulas can also be expressed by a νTL formula:

Corollary 2.38 *Let F be a closed and reduced propositional νTL formula. Then*

$$live(Mod(F)) = Mod(F \vee \neg F^\nu) = Mod(F^\nu \rightarrow F)$$

Proof: Immediate by the definition of $live(Mod(F))$ and theorem 2.36. ♠

We conclude this section by applying the preceding constructions to the property considered in example 2.6: Consider the formula $F \equiv p$ until $q \equiv \mu X.q \vee (p \wedge \circ X)$. F is a reduced formula (the reduced n-fold unfolding $G^{(n)}$ of F is easily calculated as $\circ^n F$, and F, hence also $\circ^n F$, is satisfiable), thus by theorem 2.36 its safety closure is expressed by $F^\nu \equiv \nu X.q \vee (p \wedge \circ X) \equiv p$ unless q.

Chapter 3

Axiomatization of νTL

Having defined syntax and semantics of a logic, the natural next step is to look for a formal proof system $\vdash$ in which to derive valid formulas as well as consequences of sets of given formulas (i.e., non-logical axioms), the question of interest being the soundness and adequacy of such proof systems. It is well known (see, e.g., KRÖGER [29]) that even for propositional temporal logic, complete axiom systems for the derivation of consequences from infinite sets of premises do not exist. This is due to the non-compactness of these logics, an example being the consequence $\mathcal{A} \models \Box p$ for the premise set $\mathcal{A} = \{\bigcirc^i p \mid i \in I\!N\}$. On the other hand, the consequence problem for finite axiom sets can be reduced to the problem of validity by an application of lemma 2.20, hence we will only study this problem in the following. After reviewing some known results about the axiomatization of propositional νTL in section 3.1, we will turn to the validity problem of several restrictions of first-order νTL in the subsequent sections of this chapter.

3.1 Axiomatizing propositional νTL

The validity problem of propositional νTL has been shown to be decidable in BANIEQBAL & BARRINGER [10] where a complete tableau decision procedure for propositional νTL has been given. Another tableau system for an extension of νTL by past time operators has been presented in VARDI [55]. The decidability of the original propositional μ-calculus (based on a denumerable set of successor relations) has already been established in KOZEN & PARIKH [27] using earlier work of PRATT [45]. Surprisingly, none of the decidability proofs has produced a conventional (i.e., Hilbert- or Gentzen-style) axiom system for propositional νTL. One such system for the μ-calculus, however, has been given in KOZEN [26] and proven to be complete for *aconjunctive* formulas

Axiom schemes:	
(taut)	All substitution instances of classical tautologies
(ax1)	$\neg\circ A \leftrightarrow \circ\neg A$
(ax2)	$\circ(A \wedge B) \leftrightarrow (\circ A \wedge \circ B)$
(ax3)	$\nu X.A \rightarrow A[\nu X.A/X]$
Rules:	
(mp)	$A, A \rightarrow B \ \vdash\ B$
(nex)	$A \ \vdash\ \circ A$
(ind)	$A \rightarrow B[A/X] \ \vdash\ A \rightarrow \nu X.B$

Table 3.1: Adaptation of Kozen's axiom system to νTL

(a technical condition which forbids occurrences of fixpoint formulas in certain nesting patterns). Its variant for propositional νTL (based on one single successor relation) is reproduced in table 3.1. Observe that this axiom system is sound only for deriving valid formulas, not for deriving consequences of given axioms. The soundness of the axioms and rules follows from fact 2.19, lemma 2.18 and theorem 2.22. Its completeness for full propositional νTL is still an open problem.

3.2 Sublogics of first-order νTL

It has been known for some time that, in contrast to the decidability of propositional temporal logics, their first-order counterparts are for the most part strongly incomplete, i.e., they do not admit a sound and complete formal system based on a finite (or even decidable) set of axiom schemes. Such incompleteness results have, e.g., been established for first-order temporal logics based on the operators $\circ$, $\Box$ and atnext, various semantic definitions and/or syntactic restrictions (concerning, in particular, the number and arity of function and predicate symbols of the underlying first-order language) in Szałas [48], Szałas & Holenderski [51], Abadi [2] and Kröger [30]. However, none of these proofs yielded conclusive evidence concerning the minimum requirements necessary for obtaining incompleteness — in particular, no (non-propositional) axiomatizable sublanguage of first-order temporal logic has been known, and all proofs essentially relied on the fixed interpretation of the equality predicate as the identity on the universe, leaving the situation for first-order temporal logics without identity unclear. In the remainder of this chapter, we will study the validity problem for different sublogics of first-order νTL. The underlying first-order language $\mathcal{L}$ will always be assumed to be function-free, hence the only terms of $\mathcal{L}$ are rigid and flexible variables, and the role of the first-order structure $\mathcal{S}$ underlying the semantics of first-order νTL is only to provide a universe $|\mathcal{S}|$. It will become clear that constant symbols are excluded for technical simplicity only (they will return as abbreviations later). Of course, every incompleteness result obtained for a function-free fragment immediately generalizes to full first-order νTL.

Specifically, the following two types of restrictions will be studied:

- TLP refers to temporal logics without flexible subject variables but with flexible interpretation of the predicate symbols, and without identity. Hence, only rigid subject variables are terms in such logics, and states σ_i only assign extensions $\sigma_i(P)$ to predicate symbols. $TLP^=$ denotes TLP-style logics plus identity. Additionally, the set $\mathcal{P}$ of predicate symbols contained in the underlying first-order language $\mathcal{L}$ may be fixed which is indicated by writing $TLP(\mathcal{P})$ or $TLP^=(\mathcal{P})$. In particular, $TLP(\mathcal{P}^n)$ denotes TLP logics whose languages contain at most n-place predicate symbols.

- TLV logics, on the other hand, allow flexible variables but restrict the semantics by requiring all predicate symbols to be interpreted rigidly, i.e., $\sigma_i(P) = \sigma_j(P)$ holds for every predicate symbol P and all states σ_i, σ_j occurring in the state sequence $\sigma = (\sigma_0, \sigma_1, \ldots)$ of any temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$. Again, $TLV^=$ and $TLV(\mathcal{P})$ denote, respectively, TLV logics plus equality and TLV restricted to a language containing only a fixed set of predicate symbols.

Moreover, for both TLP and TLV logics, the set of basic temporal operators may be restricted. In particular, logics based on the operators $\circ$, $\Box$ and possibly atnext instead of the fixpoint operator ν will be considered. In this case, the operators $\Box$ and atnext are no longer derived but become primitive temporal operators whose semantics are defined by the corresponding clause in fact 2.14 (in fact, $\Box F$ could be defined as $\mathsf{f}\ \mathsf{atnext}\ \neg F$). Such restrictions will also be denoted by listing the basic operators explicitly, e.g., $TLP(\mathcal{P}^1, \circ, \Box)$ denotes TLP without identity, restricted to at most unary predicate symbols and the temporal operators $\circ$ and $\Box$. The reason for considering these sublanguages is to sharpen the incompleteness results which will be obtained and tailor them to logics which have been studied in the literature. For example, KRÖGER'S [29] temporal logic can be classified as $TLV^=(\circ, \Box, \mathsf{atnext})$. BAUDINET [12], on the other hand, studies $TLP(\circ, \Box)$. Temporal operators — which we will always assume to be definable in νTL — will be represented generically as

$\mathcal{O}(F_1, \ldots, F_k)$ where the arguments indicate the subformulas on which the syntactic and semantic definitions of $\mathcal{O}$ are based. Specifically, the semantics of any such operator $\mathcal{O}$ will be assumed to be given in the form

$$\mathcal{K}^{\eta,\xi}(\mathcal{O}(F_1, \ldots, F_k)) \;=\; \mathcal{D}(\mathcal{K}^{\eta,\xi^1}(F_1), \ldots, \mathcal{K}^{\eta,\xi^k}(F_k))$$

where $\mathcal{D}$ is some (set-theoretic) expression and $\xi^1, \ldots, \xi^k$ are valuations of the propositional variables possibly differing from ξ. (Observe, however, that all subformulas are evaluated with respect to the same valuation η of rigid subject variables to prevent mixing-up of temporal and first-order aspects of the logic.)

3.3 Interpreting TLP in TLV

Clearly, the sublogics of νTL introduced in the preceding section differ in technical aspects. Sometimes, TLP logics appear more flexible and allow for technically simpler formulations of axiom sets. This is why the incompleteness proof of section 3.4 will use TLP. However, as far as the validity problem is concerned, TLP is not harder to axiomatize than TLV. Specifically, it is proven in the following lemma that for every (closed and quantifier closed) TLP-formula F not containing the equality predicate there is a TLV-formula $\tau(F)$ such that F is TLP-satisfiable (or TLP-valid) if and only if $\tau(F)$ is TLV-satisfiable (TLV-valid, respectively).

Let in the following $\mathcal{L}$ be a language for $TLP(\mathcal{O}_1, \ldots, \mathcal{O}_m)$ and define a language $\tilde{\mathcal{L}}$ for $TLV(\mathcal{O}_1, \ldots, \mathcal{O}_m)$ as follows:

- The sets $\mathcal{X}$ and $\mathcal{Y}$ of propositional and rigid subject variables, respectively, of $\tilde{\mathcal{L}}$ are the same as in $\mathcal{L}$.
- For every $n \geq 0$ such that $\mathcal{L}$ contains an n-ary predicate symbol, $\tilde{\mathcal{L}}$ contains an $(n+1)$-ary predicate symbol $Elem^n$. $\tilde{\mathcal{L}}$ contains no other predicate symbols.
- For every predicate symbol P of $\mathcal{L}$ there is a flexible variable z_P in $\tilde{\mathcal{L}}$.

Now, the formula $\tau(F)$ is inductively defined as follows:

$\tau(P(y_1, \ldots, y_n)) := Elem^n(y_1, \ldots, y_n, z_P)$ for atomic formulas $P(y_1, \ldots, y_n)$,
$\tau(X) := X$ for $X \in \mathcal{X}$, $\quad \tau(\neg F) := \neg\tau(F)$,
$\tau(F \wedge G) := \tau(F) \wedge \tau(G)$, $\quad \tau(\forall y F) := \forall y\, \tau(F)$ for $y \in \mathcal{Y}$,
$\tau(\mathcal{O}_i(F_1, \ldots, F_k)) := \mathcal{O}_i(\tau(F_1), \ldots, \tau(F_k))$ for every $\mathcal{O}_i(F_1, \ldots, F_k)$

Observe in particular that $\tau(F)$ is a closed and/or quantifier closed formula iff the same condition holds for F.

Lemma 3.1 *Let F be a closed and quantifier closed $TLP(\mathcal{O}_1, \ldots, \mathcal{O}_m)$-formula. Then F is TLP-satisfiable (TLP-valid) iff $\tau(F)$ is TLV-satisfiable (TLV-valid, respectively).*

Proof: Only the claim for satisfiability has to be proven, validity then follows by duality. Suppose there is a TLP-structure $\mathcal{K} = (\mathcal{S}, \sigma)$ such that $\mathcal{K} \models F$. We construct a TLV-structure $\tilde{\mathcal{K}} = (\tilde{\mathcal{S}}, \tilde{\sigma})$ such that $\tilde{\mathcal{K}} \models \tau(F)$.

First, define the universe $|\tilde{\mathcal{S}}| := |\mathcal{S}| \times 2^{|\mathcal{S}|^*}$, i.e., the set of pairs consisting of an element of the original universe $|\mathcal{S}|$ and a set of sequences (or tuples) of elements of $|\mathcal{S}|$. The (state-independent) interpretation of the $(n+1)$-place predicate symbol $Elem^n$ (for $n \geq 0$) in $\tilde{\sigma}$ is given by

$$(\underbrace{(d_1, M_1)}_{\in|\tilde{\mathcal{S}}|}, \ldots, \underbrace{(d_n, M_n)}_{\in|\tilde{\mathcal{S}}|}, \underbrace{(d_{n+1}, M_{n+1})}_{\in|\tilde{\mathcal{S}}|}) \in \tilde{\sigma}_i(Elem^n) \iff \underbrace{d_1 \ldots d_n}_{\in|\mathcal{S}|^*} \in M_{n+1}.$$

for every $i \geq 0$.

The valuations $\sigma_i(z_P)$ of the flexible variables z_P of $\tilde{\mathcal{L}}$ corresponding to the predicate symbols P of $\mathcal{L}$ are defined by

$$\tilde{\sigma}_i(z_P) := (d_0, \{d_1 \ldots d_n \mid (d_1, \ldots, d_n) \in \sigma_i(P)\}),$$

where $d_0 \in |\mathcal{S}| \neq \emptyset$ is an arbitrary but fixed element.

We prove that for all TLP-formulas G of the language $\mathcal{L}$, all valuations $\xi : \mathcal{X} \to I\!N$ of the propositional variables and all valuations $\eta : \mathcal{Y} \to |\mathcal{S}|$, $\tilde{\eta} : \mathcal{Y} \to |\tilde{\mathcal{S}}|$ of the rigid subject variables such that for every $y \in \mathcal{Y}$, $\eta(y) = d$ iff $\tilde{\eta}(y) = (d, M)$ for some $M \subseteq \mathcal{S}^*$ it holds that $\mathcal{K}^{\eta,\xi}(G) = \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(\tau(G))$. In particular, the assumption $\mathcal{K} \models F$ implies $\tilde{\mathcal{K}} \models \tau(F)$ since F and $\tau(F)$ are closed and quantifier closed. The proof runs by induction on the syntactical structure of G:

- $i \in \mathcal{K}^{\eta,\xi}(P(y_1, \ldots, y_n))$
  $\iff$ $(\eta(y_1), \ldots, \eta(y_n)) \in \sigma_i(P)$
  $\iff$ $\eta(y_1) \ldots \eta(y_n) \in M$ for $\tilde{\sigma}_i(z_P) = (d_0, M)$ (by def. of $\tilde{\sigma}_i(z_P)$)
  $\iff$ $(\tilde{\eta}(y_1), \ldots, \tilde{\eta}(y_n), \tilde{\sigma}_i(z_P)) \in \tilde{\sigma}_i(Elem^n)$ (by def. of $\tilde{\sigma}_i(Elem^n)$)
  $\iff$ $i \in \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(Elem^n(y_1, \ldots, y_n, z_P))$

  Observe that this proof fails for prime formulas $y_1 = y_2$, assuming $\tau(y_1 = y_2) = (y_1 = y_2)$.

- $\mathcal{K}^{\eta,\xi}(X) = \xi(X) = \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(X)$ for every propositional variable $X \in \mathcal{X}$

- $\mathcal{K}^{\eta,\xi}(\neg G) = I\!N \setminus \mathcal{K}^{\eta,\xi}(G) \overset{\text{I.H.}}{=} I\!N \setminus \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(\tau(G)) = \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(\neg\tau(G))$

- The proof for the case $G \wedge H$ is similar.
- Assume $G \equiv \mathcal{O}_i(G_1, \ldots, G_k)$ for one of the basic temporal operators $\mathcal{O}_i$ whose semantics is given by $\mathcal{D}_i(\mathcal{K}^{\eta,\xi^1}(G_1), \ldots, \mathcal{K}^{\eta,\xi^k}(G_k))$. Then the induction hypothesis may be assumed for every subexpression $\mathcal{K}^{\eta,\xi^j}(G_j)$, and we obtain

$$\begin{aligned}
& \mathcal{K}^{\eta,\xi}(\mathcal{O}_i(G_1, \ldots, G_k)) \\
= \;& \mathcal{D}_i(\mathcal{K}^{\eta,\xi^1}(G_1), \ldots, \mathcal{K}^{\eta,\xi^k}(G_k)) \\
\stackrel{\text{I.H.}}{=} \;& \mathcal{D}_i(\tilde{\mathcal{K}}^{\tilde{\eta},\xi^1}(\tau(G_1)), \ldots, \tilde{\mathcal{K}}^{\tilde{\eta},\xi^k}(\tau(G_k))) \\
= \;& \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(\mathcal{O}_i(\tau(G_1), \ldots, \tau(G_k)))
\end{aligned}$$

- $i \in \mathcal{K}^{\eta,\xi}(\forall y\, G)$

$$\begin{aligned}
\iff \;& i \in \mathcal{K}^{\eta[y/d],\xi}(G) \text{ for every } d \in |\mathcal{S}| \\
\stackrel{\text{I.H.}}{\iff} \;& i \in \tilde{\mathcal{K}}^{\tilde{\eta}[y/(d,M)],\xi}(\tau(G)) \text{ for every } d \in |\mathcal{S}|, M \subseteq |\mathcal{S}|^* \\
\iff \;& \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(\forall y\, \tau(G))
\end{aligned}$$

Conversely, assume $\tilde{\mathcal{K}} = (\tilde{\mathcal{S}}, \tilde{\sigma})$ to be any TLV-structure such that $\tilde{\mathcal{K}} \models \tau(F)$. Define a TLP-structure $\mathcal{K} = (\mathcal{S}, \sigma)$ by $|\mathcal{S}| := |\tilde{\mathcal{S}}|$ and $(d_1, \ldots, d_n) \in \sigma_i(P)$ iff $(d_1, \ldots, d_n, \tilde{\sigma}_i(z_P)) \in \tilde{\sigma}_i(Elem^n)$. Again, a simple inductive proof similar to the above establishes that $\mathcal{K}^{\eta,\xi}(G) = \tilde{\mathcal{K}}^{\eta,\xi}(\tau(G))$ for all formulas G of the language $\mathcal{L}$ and all valuations η and ξ of the rigid subject variables and the proposition variables, respectively. In particular, hence, it follows that $\mathcal{K} \models F$, proving the TLP-satisfiability of F. ♠

Notice that the lemma also holds for logical languages admitting (rigid) constant and function symbols, but not for $TLP^=$. As a consequence of lemma 3.1, strong incompleteness results for TLV can be obtained as consequences of similar results for TLP. More precisely, suppose that strong incompleteness has been established for $TLP(\mathcal{P}^n, \mathcal{O}_1, \ldots, \mathcal{O}_m)$. By the above lemma, for every TLP-valid formula F of such a language there is a TLV-valid formula $\tilde{F}$ using at most $(n+1)$-ary predicate symbols. Thus, $TLV(\mathcal{P}^{n+1}, \mathcal{O}_1, \ldots, \mathcal{O}_m)$ is also strongly incomplete. Moreover, it can easily be shown that every TLP-formula containing *at most* n-ary predicate symbols can be rewritten into an equivalent formula containing *only* n-ary predicate symbols. The TLV-formula resulting from the above translation then contains *exactly one* $(n+1)$-ary predicate symbol.

Corollary 3.2 *If the logic $TLP(\mathcal{P}^n, \mathcal{O}_1, \ldots, \mathcal{O}_m)$, for some n, does not admit a sound and complete finitistic axiomatization, then neither does $TLV(\mathcal{P}^{n+1}, \mathcal{O}_1, \ldots, \mathcal{O}_m)$ In fact, $TLV(\{P\}, \mathcal{O}_1, \ldots, \mathcal{O}_m)$ will already be incomplete for one $(n+1)$-ary predicate symbol P.*

3.4 Incompleteness of monadic TLP

In the light of corollary 3.2 we will now aim to establish the strong incompleteness of $TLP(\mathcal{P}^n)$, for some n and some choice of temporal operators. Since $TLP(\mathcal{P}^0)$, i.e. propositional νTL, is decidable, the natural candidate to study is $TLP(\mathcal{P}^1)$. As for the choice of temporal operators, logics based on either $\circ$ or $\Box$ as sole temporal operator are (equivalent to) well known modal logics (specifically, axiomatized by first-order versions of the modal systems D and $S4$, respectively, cf. Garson [20]). The logic considered in this section will be $TLP(\mathcal{P}^1, \circ, \Box)$.

One way to approach the search for an incompleteness proof is to try and find some proof appearing in the literature which can be simulated by the logic at hand. Now, Szałas & Holenderski [51] gave an incompleteness proof for $TLV^=(\mathcal{P}^0, \circ, \Box, \mathsf{atnext})$ (with flexibly interpreted proposition letters) which can be strengthened to yield an incompleteness proof of $TLV^=(\emptyset, \circ, \Box, \mathsf{atnext})$ as long as sufficiently many flexible variables are supplied: For every flexible proposition letter p appearing in the original formula, a new flexible subject variable z_p is introduced as well as an additional flexible variable z_t (intuitively representing t). Every occurrence of p in the given formula F is then replaced by $z_p = z_\mathsf{t}$, and the conjunct $\forall x(x = z_\mathsf{t} \to \Box x = z_\mathsf{t})$, enforcing rigid interpretation of z_t, is added to the formula obtained from F by this operation. A straightforward induction on the structure of F shows that this translation preserves satisfiability and validity. To conclude the incompleteness of $TLP(\mathcal{P}^1, \circ, \Box, \mathsf{atnext})$, it is possible to give a translation which takes formulas of $TLV^=(\emptyset, \circ, \Box, \mathsf{atnext})$ to $TLP(\mathcal{P}^1, \circ, \Box, \mathsf{atnext})$ while preserving validity. Such an approach has been taken in Merz [40]. However, to get rid of the atnext-operator, a direct proof has to be given, obviating the need for the translation. Nevertheless, the proof idea of Szałas & Holenderski [51] will be followed.

The idea for the elimination of atnext (and similar derived operators like unless) is to model the νTL definition of these operators in the weaker logic by introducing additional predicate symbols which simulate νTL's proposition variables and replace, say, formulas $\forall \vec{x}\vec{y}\vec{z}\, A(\vec{x}) \to (B(\vec{y})\ \mathsf{atnext}\ C(\vec{z}))$ (where $\vec{x}, \vec{y}, \vec{z}$ are tuples of rigid subject variables) by the two formulas $\forall \vec{x}\vec{y}\vec{z}\, A(\vec{x}) \to BAC(\vec{y}, \vec{z})$ and $\Box\forall\vec{y}\vec{z}(BAC(\vec{y}, \vec{z}) \to (C(\vec{z}) \to B(\vec{y})) \wedge (\neg C(\vec{z}) \to \circ BAC(\vec{y}, \vec{z})))$ where BAC is a new predicate symbol of suitable arity. However, care has to be taken to introduce unary predicate symbols only and to handle atnext-formulas in the antecedent of implications.

To simplify the presentation of the incompleteness proof, let us introduce some auxiliary notation: Assume $\mathcal{L}$ to be a first-order language containing exactly

m unary predicate symbols $P_1, \ldots, P_m$. Define the formula

$$Unique(P_k) \;\equiv\; \forall x \forall y(\Diamond(P_k(x) \wedge P_k(y)) \rightarrow \Box \bigwedge_{j=1}^{m} (P_j(x) \rightarrow P_j(y)))$$

(observe that $\Diamond$ can be defined in $TLP(\circ, \Box)$ by $\Diamond F \equiv \neg\Box\neg F$).

Intuitively, $Unique(P_k)$ states that, in any state, P_k can only be true for two elements if they are "undistinguishable" from each other by every predicate symbol of $\mathcal{L}$. This is made precise by the following lemma:

Lemma 3.3 *Let $\mathcal{L}$ contain m unary predicate symbols $P_1, \ldots, P_m$ as only predicate symbols, and $\mathcal{K} = (\mathcal{S}, \sigma)$ be a TLP-structure for $\mathcal{L}$ such that $\mathcal{K} \models Unique(P_k)$ for some $1 \leq k \leq m$. Then the relation $\sim$ defined by*

$$a \sim b \iff a = b \text{ or for some } i \in I\!N, \{a, b\} \subseteq \sigma_i(P_k)$$

is a congruence on $|\mathcal{S}|$ with respect to $P_1, \ldots, P_m$. In particular, for the structure $\tilde{\mathcal{K}} = (\tilde{\mathcal{S}}, \tilde{\sigma})$ defined by

$$|\tilde{\mathcal{S}}| = |\mathcal{S}|/\sim \;\text{ and } [d] \in \tilde{\sigma}_i(P_j) \iff d \in \sigma_i(P_j) \text{ for } 1 \leq j \leq m, d \in |\mathcal{S}|$$

(where, for $d \in |\mathcal{S}|$, $[d]$ denotes the congruence class of d modulo $\sim$) and all valuations η and ξ of the rigid subject variables and the propositional variables, respectively, it holds that $\mathcal{K}^{\eta,\xi}(F) = \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(F)$ for every $\mathcal{L}$-formula F where the valuation $\tilde{\eta}$ is defined by $\tilde{\eta}(y) = [\eta(y)]$ for every $y \in \mathcal{Y}$.

Proof: First, $\sim$ is an equivalence relation:

- $\sim$ is obviously reflexive and symmetrical.
- $a \sim b, b \sim c \;\Rightarrow\; a \sim c$:
  The only non-trivial case occurs if $a \neq b$ and $b \neq c$. But then, we have

  $$(*) \quad \{a, b\} \subseteq \sigma_i(P_k) \quad \text{and} \quad (**) \quad \{b, c\} \subseteq \sigma_j(P_k)$$

  for some $i, j \in I\!N$. Since $\mathcal{K} \models Unique(P_k)$, it follows from $(*)$ that, for every $l \geq 0$, $a \in \sigma_l(P_k)$ if $b \in \sigma_l(P_k)$ which by $(**)$ implies $\{a, c\} \subseteq \sigma_j(P_k)$, hence $a \sim c$.

For the proof that $\sim$ is a congruence relation, observe first that for $a, b \in |\mathcal{S}|$, $a \sim b$ implies $a \in \sigma_i(P_j)$ iff $b \in \sigma_i(P_j)$ for every $i \in I\!N$ and $1 \leq j \leq m$ by the definition of the formula $Unique(P_k)$. This establishes the well-definedness of the temporal structure $\tilde{\mathcal{K}}$ as well as the base case of the claim $\mathcal{K}^{\eta,\xi}(F) =$

$\tilde{\mathcal{K}}^{\tilde{\eta},\xi}(F)$. The remaining cases follows by a straightforward induction on the structure of F where only the quantifier case is given here:

$$
\begin{array}{ll}
 & i \in \mathcal{K}^{\eta,\xi}(\forall y F) \\
\iff & i \in \mathcal{K}^{\eta[a/x],\xi}(F) \text{ for every } a \in |\mathcal{S}| \\
\overset{\text{I.H.}}{\iff} & i \in \tilde{\mathcal{K}}^{\tilde{\eta}[[a]/x],\xi}(F) \text{ for every } a \in |\mathcal{S}| \\
\iff & i \in \tilde{\mathcal{K}}^{\tilde{\eta}[b/x],\xi}(F) \text{ for every } b \in |\tilde{\mathcal{S}}| \\
\iff & i \in \tilde{\mathcal{K}}^{\tilde{\eta},\xi}(\forall y F)
\end{array}
$$
♠

In other words, lemma 3.3 states that w.l.o.g. it may be assumed that for every TLP-structure $\mathcal{K} = (\mathcal{S}, \sigma)$ modelling $Unique(P_k)$ and every $i \geq 0$, $\sigma_i(P_k)$ is empty or contains exactly one element.

In the sequel, individual constants will be introduced as abbreviations for better readability, using the following convention: A "constant" c is a shorthand notation for introducing a unary predicate symbol C together with the formulas $\exists y \Box C(y)$ and $Unique(C)$. A "prime formula" $P(c)$ is an abbreviation for $\forall y(C(y) \to P(y))$. If c and d are "constants" represented by the unary predicate symbols C and D, the "formula" $c \neq d$ abbreviates $\forall y(C(y) \to \neg D(y))$.

The idea of the proof given in Szałas & Holenderski [51] is to encode the computation performed by a given deterministic Turing machine $\mathcal{T}$ on the empty tape in a formula of temporal logic. One can then give a formula which is satisfiable iff $\mathcal{T}$ enters its initial state infinitely often — but this is known to be a Σ_1^1-complete problem.

$\mathcal{T}$ is given by a finite non-empty tape alphabet $\mathcal{A}$ containing the blank symbol $\#$, a finite non-empty set Q of (machine) states including the designated initial state q_0 and final (accepting) state q_f and a set of transition rules $(q, a) \mapsto (q', a', X)$ where $q, q' \in Q$ are machine states, $a, a' \in \mathcal{A}$ are tape symbols and $X \in \{L, R\}$ tells $\mathcal{T}$ to move its head left or right. The transition rules are deterministic, i.e., for every pair (q, a) where $q \neq q_f$ there is exactly one transition rule $(q, a) \mapsto (q', a', X)$. The tape of $\mathcal{T}$ will be assumed to extend infinitely to the right side only such that there is a leftmost tape cell. Any attempt to move the head to the left of that cell will be considered illegal and cause $\mathcal{T}$ to block, i.e., stop without accepting. The logical language employed contains the individual constant ℓ representing the leftmost tape cell, an individual constant a for each tape symbol $a \in \mathcal{A}$ (in particular, the constant $\#$ will represent the blank symbol), and a constant q for each (machine) state $q \in Q$ of $\mathcal{T}$. Besides the predicate symbols representing the constants, we will use the unary predicate symbols $L, T, Q, LUE, EAL, Ext, TAL_a, ML_a, MR_a$ (for $a \in \mathcal{A}$), as well as the proposition letters $H, HBE, EBH, NoExt$, and QAH_q (for $q \in Q$). The intuitive meaning of these predicates will be introduced as they are needed.

The computation of $\mathcal{T}$ is encoded as a succession of subsequences of states where each subsequence represents a configuration (s, q, k) where $s \in \mathcal{A}^+$ is a string of tape symbols representing the tape inscriptions of the portion of the tape of $\mathcal{T}$ which has been visited during the computation, q is the current machine state of $\mathcal{T}$, and k is the index of the cell in s which is currently scanned by the head. Each state of every subsequence represents one tape cell. The formula $T(a)$ (where a is a constant representing the tape symbol $a \in \mathcal{A}$) will indicate that the cell represented by the current state is inscribed with the tape symbol a in the current configuration. The propositional variable H will be true at states representing the tape cell scanned by the head of $\mathcal{T}$ in the respective configuration. At such a state, $Q(q)$ is true for the (machine) state q of the configuration. Moreover, each tape cell of $\mathcal{T}$ is given a unique label for identification in subsequent configurations. This label will be given by the predicate L. Successive configurations will be assumed to be of non-decreasing length, hence all labels appearing in some configuration will recur in the same order in all subsequent configurations. The leftmost tape cell is labelled by ℓ, hence the formula $L(\ell)$ can serve as a delimiter between configurations. If $\mathcal{T}$ stops at some configuration (either blocking or accepting), the corresponding subsequence of states will be followed by an infinite state sequence without H becoming true.

The conjunction of the formulas introduced for the representation of constants and the following formulas constitutes the formula f_{comp} encoding the computation of $\mathcal{T}$ starting with the empty tape:

• Constants representing different tape symbols and machine states are different:

$$(3.1) \qquad \bigwedge_{a \neq a' \in \mathcal{A}} a \neq a', \qquad \bigwedge_{q \neq q' \in Q} q \neq q'$$

• Each cell carries exactly one label (modulo the congruence of lemma 3.3):

$$(3.2) \qquad Unique(L), \quad \Box \exists x L(x)$$

• Each (state representing a) cell encodes exactly one tape symbol and machine state:

$$(3.3) \qquad Unique(T), \quad \Box \bigvee_{a \in \mathcal{A}} T(a), \quad Unique(Q), \quad \Box \bigvee_{q \in Q} Q(q)$$

• The labels of all cells constituting one configuration are distinct:

$$(3.4) \qquad \Box \forall y (L(y) \rightarrow \circ LUE(y)), \Box \forall y (LUE(y) \rightarrow L(\ell) \vee (\neg L(y) \wedge \circ LUE(y)))$$

This formula is obtained by eliminating the unless-operator in $\Box\forall x(L(x) \to \circ(\neg L(x))$ unless $L(\ell))$.

• The succession of cell labels is preserved at all subsequent configurations, admitting only extensions to the right of the current configuration:

$$\Box\forall x\forall y(L(x) \wedge \circ(\neg L(\ell) \wedge L(y)) \to \Box(L(x) \leftrightarrow \circ(\neg L(\ell) \wedge L(y)))) \tag{3.5}$$

Compare this with the formula $(TAn3)$ of Kröger [30]. Note also that the formulas simulating the transitions (given below) force $L(\ell)$ at the beginning of the subsequent configuration if $\mathcal{T}$ can make a move.

• The head scans at most one tape cell of every configuration; if no cell is marked as being scanned by the head, the configuration is infinite:

$$\Box(L(\ell) \to HBE), \quad \Box(HBE \to H \vee \circ(\neg L(\ell) \wedge HBE)) \tag{3.6}$$

$$\Box(H \to \circ EBH), \quad \Box(EBH \to L(\ell) \vee (\neg H \wedge \circ EBH)) \tag{3.7}$$

These formulas are obtained by eliminating the binary operators before and unless in $\Box(L(\ell) \to H$ before $L(\ell))$ and $\Box(H \to \circ(\neg H$ unless $L(\ell)))$.

• The initial configuration is of length 1 and encodes the empty tape and (machine) state q_0:

$$L(\ell) \wedge Q(q_0) \wedge T(\#) \wedge H \wedge \circ L(\ell) \tag{3.8}$$

• Tape symbols of cells not currently scanned by the head are preserved by the transition to the next configuration. For every tape symbol $a \in \mathcal{A}$ the following formulas are introduced:

$$\Box\forall x(L(x) \wedge T(a) \wedge \neg H \to \circ TAL_a(x)) \tag{3.9}$$

$$\Box\forall x(TAL_a(x) \to (L(x) \to T(a)) \wedge (\neg L(x) \to \circ TAL_a(x))) \tag{3.10}$$

The formula $TAL_a(x)$ represents $T(a)$ atnext $L(x)$.

• $\mathcal{T}$ blocks if the head had to be moved left when it scans the leftmost tape position. For every transition $(q, a) \mapsto (q', a', L)$ of the transition table of $\mathcal{T}$ we add the following formula:

$$\Box(L(\ell) \wedge H \wedge Q(q) \wedge T(a) \to \circ\Box\neg H) \tag{3.11}$$

• Transitions $(q, a) \mapsto (q', a', L)$ of $\mathcal{T}$ are simulated by the following set of formulas which apply whenever the head is not at the leftmost tape cell. The length of the next configuration equals that of the current one:

$$\begin{array}{ll}
(3.12) & \forall x \Box(L(x) \wedge \circ(\neg L(\ell) \wedge H \wedge Q(q) \wedge T(a)) \rightarrow \\
 & \qquad \circ(ML_{a'}(x) \wedge \circ QAH_{q'} \wedge NoExt)) \\
(3.13) & \forall x \Box(ML_{a'}(x) \rightarrow (L(x) \rightarrow H \wedge \circ T(a')) \wedge (\neg L(x) \rightarrow \circ ML_{a'}(x))) \\
(3.14) & \Box(QAH_{q'} \rightarrow \circ(H \rightarrow Q(q')) \wedge \circ(\neg H \rightarrow \circ QAH_{q'})) \\
(3.15) & \Box(NoExt \rightarrow (\neg \circ L(\ell) \rightarrow \circ NoExt) \wedge \forall x(L(x) \wedge \circ L(\ell) \rightarrow \circ EAL(x))) \\
(3.16) & \forall x \Box(EAL(x) \rightarrow (L(x) \rightarrow \circ L(\ell)) \;\wedge\; (\neg L(x) \rightarrow \circ EAL(x)))
\end{array}$$

The formula $ML_{a'}(x)$ stands for "move the head to the tape cell labelled by x, and replace the tape symbol of this cell's right neighbour by a'". $QAH_{q'}$ represents the formula $\circ(Q(q')\ \mathsf{atnext}\ H)$, $NoExt$ stands for "do not extend the length of the configuration at the transition to the next one" and $EAL(x)$ for $\circ L(\ell)\ \mathsf{atnext}\ L(x)$.

• Similarly, for every transition $(q, a) \mapsto (q', a', R)$ moving the head to the right we add the following formulas. If the head scans the last cell represented in the current configuration, the number of cells is incremented by one:

$$\begin{array}{ll}
(3.17) & \forall x \Box(H \wedge Q(q) \wedge T(a) \wedge L(x) \rightarrow \\
 & \qquad (MR_{a'}(x) \wedge QAH_{q'} \wedge (\circ L(\ell) \rightarrow Ext(x)) \wedge (\circ \neg L(\ell) \rightarrow NoExt))) \\
(3.18) & \forall x \Box(MR_{a'}(x) \rightarrow \circ(L(x) \rightarrow T(a') \wedge \circ H) \wedge \circ(\neg L(x) \rightarrow MR_{a'}(x))) \\
(3.19) & \forall x \Box(Ext(x) \rightarrow \circ(L(x) \rightarrow \circ(\neg L(\ell) \wedge T(\#) \wedge \circ L(\ell))) \wedge \\
 & \qquad \circ(\neg L(x) \rightarrow Ext(x)))
\end{array}$$

The formula $MR_{a'}(x)$ plays an analogous role as $ML_{a'}(x)$ above. $Ext(x)$ stands for "extend the represented tape portion from the current configuration to the next one by one blank cell where the last cell of the current configuration is labelled by x". Notice that by the formulas 3.2 and 3.4, the new cell will receive a label which is distinct from all labels used up to the current configuration.

• $\mathcal{T}$ stops when it is in its final state q_f:

$$(3.20) \quad \Box(H \wedge Q(q_f) \rightarrow \circ\Box\neg H)$$

(For our purposes we do not have to distinguish between accepting and blocking runs of $\mathcal{T}$ but only between finite and infinite runs.)

Lemma 3.4 *If $\mathcal{K} \models f_{comp}$ for some TLP-structure $\mathcal{K} = (\mathcal{S}, \sigma)$, then the computation of $\mathcal{T}$ starting with the empty tape is encoded in the state sequence σ.*

Proof: By the formulas (3.1), (3.2) and (3.3) and the convention regarding the introduction of constants, each state σ_i encodes exactly one machine state, tape symbol and cell label (modulo the congruence relation defined in the proof of lemma 3.3). σ can be divided into subsequences $sq_0, sq_1, \ldots$ of states such that $L(\ell)$ is true at the first state of each sq_i but at no other state. The number of such subsequences sq_i is finite iff the last subsequence is of infinite length. Each subsequence sq_i of finite length which consists of the states $\sigma_0^{(i)}, \ldots, \sigma_{p^{(i)}}^{(i)}$ corresponds to a configuration $c_i = ((c_0^{(i)} \ldots c_{p^{(i)}}^{(i)}), q^{(i)}, k^{(i)})$ of $\mathcal{T}$, where $(c_0^{(i)} \ldots c_{p^{(i)}}^{(i)})$ is a string of tape symbols representing the part of the tape that has been visited during the computation up to the configuration c_i, $q^{(i)} \in Q$ is the current state of c_i and $k^{(i)}$ is the index of the cell currently scanned by the head (therefore, $0 \leq k^{(i)} \leq p^{(i)}$).

- $c_j^{(i)} \in \mathcal{A}$, for $0 \leq j \leq p^{(i)}$, is given by the (unique) tape symbol constant a, for which $a \in \sigma_j^{(i)}(T)$ (by slight misuse of notation, identifying a with the congruence class modulo $\sim$ from lemma 3.3 of those elements $d \in |\mathcal{S}|$ such that $d \in \sigma_0(A)$ where A is the unary predicate symbol representing a).
- By (3.6) and (3.7), if sq_i is finite, it contains exactly one state $\sigma_{k^{(i)}}^{(i)}$ such that $\sigma_{k^{(i)}}^{(i)}(H) = \{()\}$. The index $k^{(i)}$ gives the position of the head, and the current machine state of configuration c_i is determined by the (unique) machine state constant $q^{(i)}$ for which $q^{(i)} \in \sigma_{k^{(i)}}^{(i)}(Q)$.

By (3.8), the first subsequence sq_0 consists only of the state σ_0 and obviously encodes the start configuration of $\mathcal{T}$. Within any configuration, axiom (3.4) ensures that each cell can be identified by its unique cell label. Formula (3.5) forces labels to appear in the same order in all subsequent configurations, but allows new cells (which by (3.4) have to carry new labels) to be added at the right. From (3.9) and (3.10) it follows that all cells except possibly the one scanned by the head will keep their tape symbols at the next configuration. If the head scans the leftmost tape cell ℓ at configuration c_i and $\mathcal{T}$ had to move to the left, or if $\mathcal{T}$ signals acceptance by entering its final state, (3.11) or (3.20), respectively, force sq_{i+1} to be infinite (otherwise, (3.6) would cause it to contain a state marked by H). Since $\mathcal{T}$ was assumed to be deterministic, at each non-final configuration there is exactly one applicable transition. If $\mathcal{T}$ does not stop at configuration c_i, the formulas (3.12) – (3.19) force the subsequence sq_{i+1} to encode the configuration obtained from c_i by application of the appropriate transition. In particular, the length $p^{(i+1)}$ of sq_{i+1} is forced to equal $p^{(i)}+1$ iff $\mathcal{T}$ makes a move to the right when the head scans the rightmost cell represented in sq_i, otherwise, $p^{(i+1)} = p^{(i)}$. In case $p^{(i+1)} = p^{(i)} + 1$, (3.19) forces the newly added cell to contain a blank symbol. Thus the sequence of configurations $(c_0, c_1, \dots)$ reconstructed from the state sequence σ of $\mathcal{K}$ indeed encodes the computation of $\mathcal{T}$ starting with the empty tape. ♠

Lemma 3.5 *f_{comp} has a model $\mathcal{K}$.*

Proof: Any denumerable set $|\mathcal{S}|$ can be chosen as the universe of the temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$. The construction of the state sequence σ from the computation of $\mathcal{T}$ has been described above if the computation is infinite. If $\mathcal{T}$ stops (either accepting or blocking), the state sequence can be extended to yield a model of f_{comp} by first copying the final configuration without marking any state as scanned by the head and assigning, for the remaining states σ_i, each state a new cell label which has not occurred up to this point, defining $\sigma_i(H) = \emptyset$ for every such state and assigning an arbitrary tape symbol and machine state to every cell. ♠

As a consequence of the above lemmas 3.4 and 3.5 we obtain the following

Theorem 3.6 *The validity problem of the monadic first-order temporal logic $TLP(\mathcal{P}^1, \circ, \Box)$ is Π^1_1-complete.*

Proof: Consider the formula $f_{loop} \equiv f_{comp} \wedge \Box\Diamond(H \wedge Q(q_0))$. By the satisfiability of f_{comp} (lemma 3.5) and the method of encoding the computation of $\mathcal{T}$ in the state sequence of any model of f_{comp} (lemma 3.4), f_{loop} is satisfiable iff $\mathcal{T}$, starting on the empty tape, enters its initial state infinitely often. This Turing machine problem is known to be Σ^1_1-complete (cf. Harel, Pnueli & Stavi [22]). By duality, the validity problem of $TLP(\mathcal{P}^1, \circ, \Box)$ is Π^1_1-hard. On the other hand, it is a consequence of the infinitary axiomatization of $TLV(\circ, \Box)$ provided in Szałas [49] (using lemma 3.1) that the validity problem for $TLP(\circ, \Box)$ — even without the restriction to monadic predicate symbols — is contained in Π^1_1. ♠

With the aid of corollary 3.2, the incompleteness results of Szałas & Holenderski [51] and also of Kröger [30] for TLV logics can now be strengthened: It is not necessary to have logics with identity for TLV to be incomplete; any (uninterpreted) binary predicate symbol suffices.

Corollary 3.7 *The validity problem for $TLV(\{P\}, \circ, \Box)$ where P is a binary predicate symbol is Π^1_1-complete.*

In the next section the validity problem for $TLV(\mathcal{P}^1)$, without restrictions on the temporal operators, will be shown to be decidable, rendering the condition of corollary 3.7 regarding the signature of TLV-languages not only sufficient but also necessary for obtaining strong incompleteness.

3.5 Decidability of pure monadic TLV

In contrast to the incompleteness results for $TLP(\mathcal{P}^1)$ it will be shown here that the validity problem of monadic TLV-formulas (without function symbols and equality) is decidable — even if all temporal operators definable by fixpoint formulas are admitted. The proof idea of the corresponding result for classical first-order logic (see, e.g., Ackermann [6]) carries over to $TLV(\mathcal{P}^1)$. Notice, however, that in classical first-order logic, adding identity to an otherwise monadic signature preserves decidability in contrast to the incompleteness result for $TLV^{=}(\emptyset)$ following from Szałas & Holenderski [51] by the above remark. The proof will be divided into two parts: First, we show that a $TLV(\mathcal{P}^1)$-formula F is valid iff F holds in all structures with universes of a certain finite cardinality which depends on the number of distinct predicate symbols appearing in F. The validity of $TLV(\mathcal{P}^1)$-formulas in structures whose universes are of a given finite cardinality will in a second step be reduced to the validity of a formula of propositional νTL which is decidable by the results cited in section 3.1.

Lemma 3.8 *Let F be a closed and quantifier closed formula of $TLV(\mathcal{P}^1)$ containing p unary predicate symbols $P_1, \ldots, P_p$. F is valid iff $\mathcal{K} \models F$ for every TLV-structure $\mathcal{K} = (\mathcal{S}, \sigma)$ whose universe $|\mathcal{S}|$ contains at most 2^p elements.*

Proof: We only have to prove the "if"-part. Assume F to be valid in every TLV-structures whose universe contains at most 2^p elements and let $\mathcal{K} = (\mathcal{S}, \sigma)$ be *any* $TLV(\mathcal{P}^0 \cup \{P_1, \ldots, P_p\})$-structure. Define an equivalence relation $\sim$ on $|\mathcal{S}|$ by

$$a \sim b \quad \Longleftrightarrow \quad a \in \sigma_0(P_i) \text{ iff } b \in \sigma_0(P_i) \quad \text{for all } 1 \leq i \leq p,$$

i.e., $a \sim b$ iff a and b are undistinguishable by all predicates assigned to $P_1, \ldots, P_p$ in $\mathcal{K}$ (recall that all predicate symbols are interpreted rigidly in TLV-structures). The relation $\sim$ induces at most 2^p equivalence classes on $|\mathcal{S}|$, each class being characterized by the subset of predicates holding of its members. Now, define the TLV-structure $\tilde{\mathcal{K}} = (\tilde{\mathcal{S}}, \tilde{\sigma})$ by

$|\tilde{\mathcal{S}}| := |\mathcal{S}|/\sim$,
$[d] \in \tilde{\sigma}_i(P_j)$ iff $d \in \sigma_i(P_j)$ for $d \in |\tilde{\mathcal{S}}|, i \geq 0$ and $1 \leq j \leq p$,
$\tilde{\sigma}_i(Q) := \sigma_i(Q)$ for every proposition letter Q, and
$\tilde{\sigma}_i(z) := [\sigma_i(z)]$ for $i \geq 0$ and $z \in \mathcal{Z}$

Observe that $\tilde{\sigma}_i(P_j)$ is well-defined because of $d \in \sigma_i(P_j)$ iff $d' \in \sigma_i(P_j)$ for $d \sim d'$.

Since $|\tilde{\mathcal{S}}|$ contains at most 2^p elements (viz. the equivalence classes of $\sim$ on $|\mathcal{S}|$), by the assumption it follows that $\tilde{\mathcal{K}} \models F$. A straightforward structural induction shows that for all $TLV(P_1, \ldots, P_p)$-formulas G and all valuations ξ and η of the proposition variables and the rigid subject variables, $\tilde{\mathcal{K}}^{\tilde{\eta},\xi}(G) = \mathcal{K}^{\eta,\xi}(G)$ if $\tilde{\eta}(y) = [\eta(y)]$. In particular, since F is closed and quantifier closed, we obtain $\mathcal{K} \models F$ and, since $\mathcal{K}$ was arbitrary, $\models F$. Notice that the boundary for the size of the universes to be considered does not depend on the number of proposition letters contained in F because the universe $|\mathcal{S}|$ is irrelevant for their interpretation. ♠

The number of structures to inspect for validity can be reduced further: Every formula F which is valid in all temporal structures $\mathcal{K} = (\mathcal{S}, \sigma)$ whose universe $|\mathcal{S}|$ is of some fixed finite cardinality n is clearly also valid in all structures $\mathcal{K}' = (\mathcal{S}', \sigma')$ whose universe is of cardinality $0 < m \leq n$: We may add $n - m$ elements which are indistinguishable from some arbitrary element of $|\mathcal{S}'|$ by all predicates of the language without affecting the truth value of F in any state of $\mathcal{K}'$.

Therefore, the validity problem of $TLV(\{P_1, \ldots, P_p\})$ reduces to checking validity in all structures whose universe has cardinality 2^p. This problem can be encoded in propositional νTL (in the sequel referred to as PTL) and is therefore decidable.

Lemma 3.9 *For every closed and quantifier closed formula F of $TLV(\mathcal{P}^1)$ and every natural number n, there is a formula F' of propositional νTL which is PTL-valid (resp., PTL-satisfiable) iff F is TLV-valid (resp., TLV-satisfiable) in TLV-structures whose universe has cardinality n.*

Proof: Again, only the claim concerning satisfiability is proven, since the claim for validity follows by duality. Assume, for technical convenience, all bound rigid subject variables appearing in F to be distinct. Furthermore, w.l.o.g., the universe of all relevant TLV-structures may be fixed as $|\mathcal{S}| = \{1, \ldots, n\}$. Suppose F contains p unary predicate symbols $P_1, \ldots, P_p$, q proposition letters $Q_1, \ldots, Q_q$ and m distinct flexible variables $z_1, \ldots, z_m$. Let the language of propositional logic contain, in addition to $Q_1, \ldots, Q_q$, $p * n$ proposition letters $P_{11}, \ldots, P_{1n}, \ldots, P_{p1}, \ldots, P_{pn}$ and $m * n$ proposition letters $Z_{11}, \ldots, Z_{1n}, \ldots, Z_{m1}, \ldots, Z_{mn}$. The formula F' is defined as the conjunction $F' \equiv P \wedge Z \wedge \gamma(F)$ of the following PTL-formulas:

$$
\begin{aligned}
P &\equiv \bigwedge_{j=1}^{p} \bigwedge_{k=1}^{n} (\Diamond P_{jk} \rightarrow \Box P_{jk}) \wedge \bigwedge_{j=1}^{q} (\Diamond Q_j \rightarrow \Box Q_j), \\
Z &\equiv \Box \bigwedge_{j=1}^{m} (\bigvee_{k=1}^{n} Z_{jk} \wedge \bigwedge_{k=1}^{n} (Z_{jk} \rightarrow \bigwedge_{l \neq k} \neg Z_{jl}))
\end{aligned}
$$

Intuitively, P_{jk} will encode if $k \in \sigma_0(P_j)$. The formula P simulates rigid interpretation of all predicate symbols in TLV.[1] The truth of Z_{jk} in the i-th state of the PTL structure will denote that $z_j = k \in |\mathcal{S}|$ in the corresponding state of the TLV structure. The formula Z therefore expresses the condition that z_j is assigned exactly one value at every state.

The remaining conjunct $\gamma(F)$ is inductively defined according to the outermost operator of F, where $G[k/y]$, for $1 \leq k \leq n$, denotes the string resulting from G by replacing all occurences of the variable y by the number k:

$$\gamma(\forall y G) := \gamma(G[1/y]) \wedge \ldots \wedge \gamma(G[n/y])$$
$$\gamma(\neg G) := \neg\gamma(G), \qquad \gamma(G \wedge H) := \gamma(G) \wedge \gamma(H), \quad \gamma(X) := X \text{ for } X \in \mathcal{X}$$
$$\gamma(\circ G) := \circ\gamma(G), \qquad \gamma(\nu X.G) := \nu X.\gamma(G)$$
$$\gamma(P_j(k)) := P_{jk}, \qquad \gamma(P_j(z_k)) := \bigvee_{l=1}^{n}(Z_{kl} \wedge P_{jl}), \quad \gamma(Q_j) := Q_j$$

Notice that $\gamma(F)$ is indeed a closed formula of propositional temporal logic since F was assumed to be closed and quantifier closed.

Let us call a PTL-structure $\tilde{\mathcal{K}} = \tilde{\sigma}$ and a TLV-structure $\mathcal{K} = (\mathcal{S}, \sigma)$ (with universe $|\mathcal{S}| = \{1, \ldots, n\}$) *adjoined* if they satisfy the following conditions:

$$\tilde{\mathcal{K}} \models P \wedge Z, \quad \sigma_0(Q_j) = \tilde{\sigma}_0(Q_j) \quad (\text{for } 1 \leq j \leq q),$$
$$k \in \sigma_0(P_j) \text{ iff } \tilde{\sigma}_0(P_{jk}) = \{()\} \quad \text{for } 1 \leq j \leq p \text{ and } 1 \leq k \leq n, \quad \text{and}$$
$$\sigma_i(z_j) = k \text{ iff } \tilde{\sigma}_i(Z_{jk}) = \{()\} \quad \text{for } i \in \mathbb{N}, 1 \leq j \leq m \text{ and } 1 \leq k \leq n$$

It is easily checked that for every TLV-structure $\mathcal{K}$ there is an adjoined PTL-structure $\tilde{\mathcal{K}}$ and, conversely, that for every PTL-structure satisfying $P \wedge Z$, there is an adjoined TLV-structure. Moreover, if $\mathcal{K}$ and $\tilde{\mathcal{K}}$ are adjoined TLV- and PTL-structures, respectively, then for every TLV-formula G and all valuations η and ξ it holds that $\mathcal{K}^{\eta,\xi}(G) = \tilde{\mathcal{K}}^{\xi}(\gamma(\eta(G)))$, where $\eta(G)$ results from G by replacing every free rigid variable y by $\eta(y) \in |\mathcal{S}|$:

First, note that from $\tilde{\mathcal{K}} \models P$ it follows that for every $i \in \mathbb{N}$, $\tilde{\sigma}_i(P_{jk}) = \tilde{\sigma}_0(P_{jk})$ for $1 \leq j \leq p$ and $1 \leq k \leq n$, and $\tilde{\sigma}_i(Q_j) = \tilde{\sigma}_0(Q_j)$ for $1 \leq j \leq q$. Since predicate symbols are interpreted rigidly in TLV structures, similarly $\sigma_i(P_j) = \sigma_0(P_j)$ for $1 \leq j \leq p$ and $\sigma_i(Q_j) = \sigma_0(Q_j)$ for $1 \leq j \leq q$.

- $G \equiv Q_j$:

  $i \in \mathcal{K}^{\eta,\xi}(Q_j) \iff \sigma_i(Q_j) = \{()\} \iff \sigma_0(Q_j) = \{()\} \iff$
  $\tilde{\sigma}_0(Q_j) = \{()\} \iff \tilde{\sigma}_i(Q_j) = \{()\} \iff i \in \tilde{\mathcal{K}}^{\xi}(Q_j)$

[1] If TLV admitted flexible interpretations of proposition letters, the only change in this proof would be to drop the conjunct enforcing strict interpretation of the Q_j's. Thus, flexible proposition letters may be added to monadic TLV without affecting its decidability.

- $G \equiv P_j(y)$: Let $\eta(y) = k$.
  $i \in \mathcal{K}^{\eta,\xi}(P_j(y)) \iff k \in \sigma_i(P_j) \iff k \in \sigma_0(P_j) \iff$
  $\tilde{\sigma}_0(P_{jk}) = \{()\} \iff \tilde{\sigma}_i(P_{jk}) = \{()\} \iff i \in \tilde{\mathcal{K}}^{\xi}(\gamma(P_j(k)))$

- $G \equiv P_j(z_k)$:
  $i \in \mathcal{K}^{\eta,\xi}(P_j(z_k)) \iff \sigma_i(z_k) \in \sigma_i(P_j) \iff \sigma_i(z_k) \in \sigma_0(P_j)$
  $\iff$ for some $1 \le l \le n$, $\tilde{\sigma}_i(Z_{kl}) = \{()\}$ and $\tilde{\sigma}_i(P_{jl}) = \tilde{\sigma}_0(P_{jl}) = \{()\}$
  $\iff i \in \tilde{\mathcal{K}}^{\xi}(\bigvee_{l=1}^{n}(Z_{kl} \wedge P_{jl}))$

- $G \equiv X \in \mathcal{X}$:
  $\mathcal{K}^{\eta,\xi}(X) = \xi(X) = \tilde{\mathcal{K}}^{\xi}(X)$

- $G \equiv \neg H$, $G \equiv \circ H$, $G \equiv H \wedge I$: apply the induction hypothesis.

- $G \equiv \nu X.H$:
  $\mathcal{K}^{\eta,\xi}(\nu X.H) = \bigcup\{I \subseteq I\!N \mid I \subseteq \mathcal{K}^{\eta,\xi[I/X]}(H)\}$
  $\stackrel{\text{I.H.}}{=} \bigcup\{I \subseteq I\!N \mid I \subseteq \tilde{\mathcal{K}}^{\xi[I/X]}(\gamma(\eta(H)))\}$
  $= \tilde{\mathcal{K}}^{\xi}(\nu X.\gamma(\eta(H)))$

- $G \equiv \forall y H$:
  $\mathcal{K}^{\eta,\xi}(\forall y H) = \bigcap_{d \in |\mathcal{S}|} \mathcal{K}^{\eta[d/y],\xi}(H) = \mathcal{K}^{\eta[1/y],\xi}(H) \cap \ldots \cap \mathcal{K}^{\eta[n/y],\xi}(H)$
  $\stackrel{\text{I.H.}}{=} \tilde{\mathcal{K}}^{\xi}(\gamma(\eta[1/y](H))) \cap \ldots \cap \tilde{\mathcal{K}}^{\xi}(\gamma(\eta[n/y](H)))$
  $= \tilde{\mathcal{K}}^{\xi}(\gamma(\eta(H[1/y]))) \cap \ldots \cap \tilde{\mathcal{K}}^{\xi}(\gamma(\eta(H[n/y])))$
  $= \tilde{\mathcal{K}}^{\xi}(\gamma(\eta(\forall y\, H)))$

In particular, F is TLV-satisfiable (over the n-element universe $|\mathcal{S}|$) iff $F' \equiv P \wedge Z \wedge \gamma(F)$ is PTL-satisfiable and vice versa. ♠

The two preceding lemmas, together with the decidability of propositional νTL, yield the decidability result for $TLV(\mathcal{P}^1)$:

Theorem 3.10 *The validity problem of pure monadic first-order νTL with rigid interpretation of the predicate symbols is decidable if the underlying first-order language does not contain function symbols.*

Observe that individual constants as well as flexible interpretation of 0-place predicate symbols (proposition letters) could be admitted without destroying decidability. Moreover, the same proof method generalizes to past-time temporal operators (see, e.g., LICHTENSTEIN, PNUELI & ZUCK [34]).

To summarize the results concerning the validity problem of νTL (and its sublogics), cf. table 3.2. All results hold uniformly for logics based on the temporal operators $\circ$ and $\Box$ only as well as for full νTL. Recently, SZAŁAS [50] gave an infinitary axiomatization of the sublogic of first-order νTL consisting of those formulas F which induce *continuous* mappings τ_F, establishing containment in Π^1_1 of the validity problem of such formulas.

logic	maximum arity of predicate symbols		
	0	1	≥ 2
TLV	decidable	decidable	non-axiomatizable
$TLV^=$	non-axiomatizable[2]	non-axiomatizable	non-axiomatizable
TLP	decidable	non-axiomatizable	non-axiomatizable
$TLP^=$	decidable	non-axiomatizable	non-axiomatizable

Table 3.2: Validity problem of different temporal logics

[2]Even $TLV^=(\emptyset)$ can be shown to be non-axiomatizable by strengthening the proof given in SZAŁAS & HOLENDERSKI [51] along the lines given in the remark at the beginning of section 3.4.

Chapter 4

Propositional νTL programs

4.1 A model for reactive systems

It has been noted in the introduction that the behavior of reactive systems is described as an interaction of the system and its environment. This section will discuss the specification and development of reactive systems using the logic νTL and motivate the principal decisions underlying the restrictions on "executable" νTL formulas. A reactive system in general consists of several *modules* working concurrently and cooperating in producing the behavior of the entire system (cf. figure 4.1). A complete description of the state of the system is therefore given as a combination of the state of the environment and the states of each module. The state of one single module is given by listing the values of the module's *state components* (such as variables, channels or conditions the module waits for before performing some action, depending on the basic entities offered by the programming model). Only a subset of the state components of each module and of the entire system will be visible to other modules or to the system's environment, hence it may be distinguished between the *observable state* (given by the values of the *interface components*) and the *complete state* which also includes values of *internal state components* (such internal components are not displayed in figure 4.1). The interface components of the entire system will be a subset of the interface components of the single modules the system consists of. For example, in the reactive system of figure 4.1, the interface components z of module M_2 and w of module M_3 are also interface components of the system relative to the environment (indicated by the arrows traversing the interface which is represented by the oval line enclosing the system). In contrast, the values of x and y cannot be observed by the environment; they are private to the modules M_1 and M_2. The environment presents its inputs to the system by assigning certain values

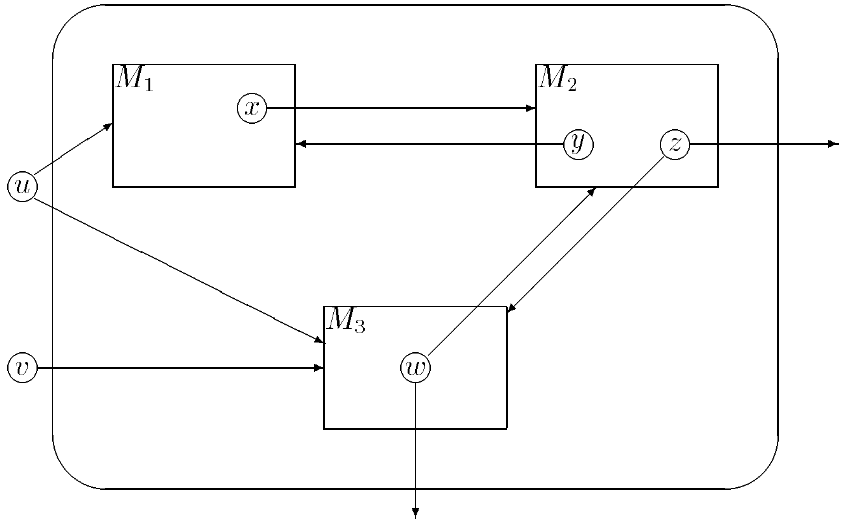


Figure 4.1: A simple reactive system

to its interface components (u and v in the example). Thus, system and environment are fully symmetrical: What is considered as the environment in one specification may be described as a reactive system in another specification with the reactive system of the former specification as its environment. Notice, moreover, that each module of a reactive system can in turn be viewed as a reactive system whose environment consists of (the relevant portion of) the system's environment plus the interface components of other modules of the system it interacts with. For example, the environment of module M_1 of figure 4.1 consists of the environment component u and the interface component y of module M_2.

Besides describing its module structure, a specification of a reactive system has to characterize the set of its desired behaviors, i.e. the responses (observed as values of the system's interface components) to inputs presented to the system as values assumed by the environment's interface components. A specification thus identifies a list of properties the system ought to possess. This "declarative" approach leads to modular specifications: Additional requirements can be introduced by adding (i.e. conjoining) a new property to the already existing specification. Similarly, the specification can be changed by dropping or modifying some properties contained in the list. Although the environment of a reactive system neither can nor should be determined by the system's specification and must therefore not be restricted by it (cf. the studies of *realizability* in Abadi, Lamport & Wolper [4] and Pnueli & Rosner [44]), frequently the system designer will have some knowledge about the behavior of (at least parts of) the environment. Therefore, he or she should not be restricted from stating this knowledge in the form of *environment assumptions*, requiring the system to behave properly only as long as the environment observes the assumptions placed on its behavior.

In the following, the temporal logic νTL will be used as a specification language. This chapter only deals with finite-state systems, which can be specified using propositional temporal logics, encoding the finite set of values each state component may assume by (lists of) boolean variables which can be represented in propositional νTL by proposition letters. First-order specifications for potentially infinite-state systems will be considered in chapter 5. The development of a reactive system starts from an initial specification and proceeds by applying *refinement steps*, obtaining intermediate specifications until a description of the system is reached which is sufficiently concrete to allow an *implementation* of this final specification in some suitable programming language. Two main types of refinement steps can be distinguished in this process:

- *Programming in the large (modularization)*: The system (or some module) is decomposed in several (sub-)modules. In particular, the set of interface components of each (sub-)module has to be identified, and properties of the (sub-)modules have to be stated such that the concurrent composition of the new modules satisfies the previous specification.
- *Programming in the small*: A single module's specification is refined by requiring additional properties or replacing properties of the previous specification by new ones. In general, it is required that the new specification does not permit behaviors which were not admissible for the previous one. Stated differently, for any specification S' obtained by applying a refinement step to a specification S, it should hold that S' implies S. Moreover, the new specification should be "closer at the implementation level" in some sense. This concept should be given a more precise meaning by the development method, possibly using syntactical criteria, in order to give some guidance to the developer.

Modularization will be the subject of chapter 6; this chapter and also chapter 5 will only deal with the development of one single module. The final step in system development then consists in an implementation of every module in some programming language, accompanied by a proof that this implementation indeed satisfies the final specification. It has frequently been recognized as a drawback of such an approach to program development (cf., for example, Kröger [28]) that the developer has to master several different formalisms, viz. the specification language together with its associated proof methods, the programming language, and a verification formalism for the proof that the resulting program satisfies the final specification. Also, whereas numerous methodologies for the refinement of specifications of reactive systems have been proposed, the transition from the final specification to the implementation is, in many cases, quite unclear, and left to the intuition of the programmer. No new methodology for program development will be proposed here, but we will aim to give precise criteria for "implementable" specifications, essentially by eliminating the implementation step: A class of temporal logic formulas will be identified using purely syntactical criteria which can be "executed" directly by constructing a model for a given set of such formulas. In fact, such an "interpreter" algorithm which actually performs the model construction will explicitly be provided (cf. algorithm 4.32). Of course, to make the model construction viable, the satisfiability of the formulas making up a specification has to be ensured. In fact, more is required for the successful execution of the final specification (or program) as a reactive system: For every behavior of the environment which is admissible according to the environment assumptions, the interpreter algorithm should be able to construct a model of the specification. It will be possible to express this requirement in the form of proof obligations for the developer. This approach allows the programmer to use the same formalism throughout the entire development process since both specifications and programs are expressed as sets of νTL formulas, and all proofs which become necessary during the system development are carried out in the same formalism. Notice that similar ideas — unifying the languages and obtaining executable specifications — have also contributed to the development of logic programming languages like PROLOG (cf. KOWALSKI [25]). However, classical logic programming languages are based on the "proofs as programs"-paradigm where the underlying theory or the (standard) model is viewed as a static object, and queries to this theory (or model) are processed by searching a proof of the query. This search, if successful, yields some variable bindings representing the result of the computation as a side effect. In contrast, for the execution of νTL programs we will view formulas *imperatively* (a term introduced in GABBAY [17]), dynamically generating models.

Let us first state what we consider to be necessary prerequisites for any programming language for distributed reactive systems:

- *Incremental construction of runs*: The execution of every program proceeds in discrete steps, starting from an initial state and reacting to inputs from the environment according to the directions given by the operational semantics of the program. Although a run of the system is by definition an inifinite object, every single reaction of the system must be based on a finite sequence of environment actions and must be output after a finite number of steps.

- *Irrevocability of previous outputs*: Like a chess player is not allowed to change his draws, the system must not revoke any reactions to environment inputs after they may have been observed by the environment (i.e. after interface state components have been assigned values reflecting the system's reactions). Otherwise, the environment would not know if the system's observed reactions will remain stable or if they will change again. This condition precludes (observable) backtracking by the system. Notice that conventional logic programming languages also do not exhibit this type of backtracking. In PROLOG, e.g., backtracking may well occur, but only before computation results are displayed to the user.

- *Absence of essential non-determinism*: If no (observable) backtracking is allowed, then no alternative open to the program at some point may eventually lead to a "dead end" from which no further execution is possible if at least one choice taken by the program would have permitted a complete run of the system given identical environment behavior. In other words, all possibly decisive choices in the sense that one alternative may lead to a successful execution whereas another one leads to failure have to be resolved by the programmer.

From these general principles we may deduce some restrictions concerning the syntactic form of νTL formulas admissible for direct execution (called *program clauses*):

- Essentially, program clauses will be implications of the form

  $$Present \rightarrow Future,$$

  where *Present* is a state formula, and *Future* is again a program clause. This restriction ensures that the "firing" of no clause depends on future behavior of the environment or the system. Whatever is needed of the behavior history up to the present point has to be recorded in the present state of the system. If past-time temporal operators were present in νTL, they could be allowed to appear in the antecedent of program clauses. The restriction of antecedents to state formulas allows the interpreter to operate more efficiently, both in terms of space and time complexity. Moreover, it follows from the results on expressive completeness for the $(\circ, \mathsf{atnext})$-fragment of νTL (Gabbay et al. [18]) and full νTL (cf. Thomas [54] and Stirling [47]) w.r.t. the monadic first-order and second-order theories, respectively, of linear orders over ω-models that no expressive power is lost by imposing this restriction.

- The *Future* part of every (sub-)clause must not contain disjunctions or least fixpoint (μ-) operators. By theorem 2.30, it will therefore express a safety property. Any disjunction appearing in a (sub-)clause represents a choice left to the program and must therefore be non-essential in the sense of the general principle stated above. Hence, there is no point in leaving the choice to the interpreter. Rather, it should be taken before by the programmer. This restriction is not strictly essential for the execution of programs but it will simplify proof obligations during the development of a program (cf. the notion of compatible programs introduced in definition 4.31 below). Notice that a distinction is made between disjunctions and implications (notwithstanding the classical framework underlying the semantic definitions of propositional connectives in νTL): The antecedent of any implication occurring in a program clause is viewed as a condition to be tested (not falsified!) by the interpreter. Its truth value in some state is not controlled by the program clause. The succedent, on the other hand, is viewed imperatively, requiring appropriate action of the interpreter to make it true in the relevant states of the model constructed. This is why disjunctions in the consequent parts of program clauses, unlike implications, would present a choice to the interpreter. Similarly, if formulas with non-trivial liveness extensions were allowed as program clauses, there would always be a (local) choice to delay the liveness obligation. However, an interpreter always taking this alternative would not construct a model of the formula. Thus, formulas (essentially) containing the μ-operator violate the above principle forbidding essential non-determinism. On the other hand, imposing a strategy for satisfying liveness requirements such as trying to fulfill them whenever they do not contradict other program rules would place too strong a proof obligation on the interpreter — if it could resolve the decision at all (cf. the discussion in Barringer et al [11] on this subject). Besides, the operational semantics of the liveness requirement under such an execution strategy does not agree with the original formula but rather corresponds to another formula expressing a safety property.

- Program clauses may only assign values to module components. This will be ensured by allowing only proposition letters representing module components to appear in consequent parts of program clauses.

Let us first define the class of *extended program clauses* (of propositional νTL). This class of formulas, although not in general acceptable as input to the model construction algorithm, will play an important role in the following discussions. Additional restrictions imposed on (proper) *program clauses* which form the class of "executable" formulas will be introduced below. Assume a given set $\mathcal{V}$ of proposition letters.

Definition 4.1 (extended program clauses)

- *Proposition letters* $p \in \mathcal{V}$ *and their negations as well as the propositional constants* t *and* f *are* literals.
- *Every literal is an* antecedent. *If* A *and* B *are antecedents, then* $A \wedge B$ *and* $A \vee B$ *also are antecedents.*
- *Every literal as well as every proposition variable* $X \in \mathcal{X}$ *is a* pre-clause. *If* A *is an antecedent and* P, Q *are pre-clauses, then* $P \wedge Q$, $\circ P$, $A \rightarrow P$, *and* $\nu X.P$ *for any variable* $X \in \mathcal{X}$ *all of whose free occurrences in* P *are in the scope of at least one* $\circ$*-operator, are pre-clauses.*
- *Every closed pre-clause is an* extended program clause.

Hence, formulas like

$$p \vee q \rightarrow r \wedge s,$$
$$\begin{aligned} \Box(p \rightarrow \circ q) &\equiv \nu X.(p \rightarrow \circ q) \wedge \circ X \quad \text{or} \\ \Box(p \rightarrow q \ \mathsf{unless}\ r) &\equiv \nu X.(p \rightarrow (\nu Y.\neg r \rightarrow q \wedge \circ Y)) \wedge \circ X \end{aligned}$$

are extended program clauses, whereas

$$r \wedge s \rightarrow p \vee q,$$
$$\begin{aligned} \Box(\circ q \rightarrow p) &\equiv \nu X.(\circ q \rightarrow p) \wedge \circ X \quad \text{or} \\ \Box(p \rightarrow q \ \mathsf{until}\ r) &\equiv \nu X.(p \rightarrow (\mu Y.\neg r \rightarrow q \wedge \circ Y)) \wedge \circ X \end{aligned}$$

are not. Notice that by lemma 2.26 the condition that all free occurrences of X in every pre-clause $\nu X.F$ are in the scope of at least one $\circ$ operator is no restriction on the expressiveness of extended program clauses.

For proper program clauses, some additional syntactical restrictions will be introduced: First, assume the set $\mathcal{V}$ of proposition letters to be partitioned into a set $\mathcal{V}_\Pi$ representing state components of the module (module components) and a set $\mathcal{V}_E$ representing environment components. As indicated above, only literals built from module components may appear in the consequent parts of pre-clauses; environment components may only appear in antecedents. It will be permitted for program clauses to have an effect on defining the current state, i.e. "the future is reflexive". (In contrast, METATEM [11] separates the time instants referred to by antecedents and consequent parts by requiring antecedents to be pure past-time formulas). However, mutual or cyclic dependencies between module components such as in $p \rightarrow p$ or $\Box(p \rightarrow \neg q) \wedge (r \rightarrow p \ \mathsf{atnext}\ q)$ should be avoided because such kinds of dependencies are prone to introducing contradictions (witness the second example). In any case, it is difficult to define an imperative meaning of formulas in their presence (this will become still more evident in the first-order case). To avoid such undesired effects, the program developer has to define the *dependency order* as a partial order $<$ on the set $\mathcal{V}_\Pi$ of module components. This order is extended to arbitrary literals $L_1 \equiv (\neg)p$, $L_2 \equiv (\neg)q$ by defining

$$L_1 < L_2 \quad \text{if} \quad \begin{array}{l} p, q \in \mathcal{V}_\Pi \text{ and } p < q \quad \text{or} \\ p \in \mathcal{V}_E \cup \{\mathsf{t}, \mathsf{f}\} \text{ and } q \in \mathcal{V}_\Pi. \end{array}$$

Formally, program clauses P and antecedents A for program clauses are defined inductively together with sets $read(A)$ and $write(P)$ of state components which are "read" by A and "written" by P, respectively, in the present state:

Definition 4.2 (program clauses)

- *Every literal $L \equiv (\neg)p$ is an* antecedent *where $read(L) = \{p\}$ if $p \in \mathcal{V}$ is a proposition letter, and $read(L) = \emptyset$ if $p \in \{\mathsf{t}, \mathsf{f}\}$.*
- *If A and B are antecedents then $A \wedge B$ and $A \vee B$ also are antecedents where $read(A \wedge B) = read(A \vee B) = read(A) \cup read(B)$.*
- *For every proposition letter $p \in \mathcal{V}_\Pi$, p and $\neg p$ are* pre-clauses *where $write(p) = write(\neg p) = \{p\}$.*
- *If P is a pre-clause then so is $\circ P$ where $write(\circ P) = \emptyset$. Moreover, if $X \in \mathcal{X}$ is a proposition variable then $\circ X$ is a pre-clause where again $write(\circ X) = \emptyset$.*
- *If P is a pre-clause and X is a proposition variable then $\nu X.P$ is a pre-clause where $write(\nu X.P) = write(P)$.*
- *If P and Q are pre-clauses then $P \wedge Q$ is a pre-clause where $write(P \wedge Q) = write(P) \cup write(Q)$.*
- *If A is an antecedent and P is a pre-clause such that $p < q$ holds for all $p \in read(A), q \in write(P)$, then $A \to P$ is a pre-clause where $write(A \to P) = write(P)$.*
- *Every closed pre-clause is a* program clause.

Observe, in particular, that every program clause is also an extended program clause. All the examples for extended program clauses given above can also be turned into examples for (proper) program clauses by defining a suitable partial order on the proposition letters. For example, for $\varphi \equiv \Box(p \to q \ \mathsf{unless}\ r)$ to be a program clause it has to hold that $p < q$ and $r < q$ since r and p appear in antecedents of sub-pre-clauses ψ of φ where $q \in \mathit{write}(\psi)$ (cf. the explicit νTL definition of φ above). On the other hand, no order can be defined such that $\Box(p \rightarrow \neg q) \wedge (r \rightarrow p \ \mathsf{atnext}\ q)$ becomes an admissible program clause since both $p < q$ and $q < p$ would have to hold. Observe that the dependency order $<$ reflects requirements necessary for defining an operational semantics of program clauses but has no logical meaning: The logically equivalent formula $\Box(q \rightarrow \neg p) \wedge (r \rightarrow p \ \mathsf{atnext}\ q)$ is an acceptable program clause (assuming $r < p$ and $q < p$). The dependency order will guide the model construction algorithm in finding a sequential order in which to define the valuation of proposition letters in every state.

4.2 Transition graphs

As a convenient means for the definition of an operational semantics complementing the declarative logical semantics of (extended) program clauses, this class of formulas will be represented in an automata-theoretic framework. The necessary technical definitions and some basic lemmas relating these automata and extended program clauses will be studied in this section.

We will define *transition graphs* which are essentially finite automata with two different types of edges whose nodes as well as the edges are inscribed with (sets of) state formulas. Assume, as before, some given set $\mathcal{V}$ of proposition letters.

Definition 4.3 (transition graphs)

A transition graph $T = (N, I, S, \vartheta, \zeta, n_0)$ *consists of:*

- *a finite non-empty set N of* nodes*,*
- *an acyclic relation $I \subseteq N \times N$ representing* internal edges*, i.e. for no $n \in N$, it holds that $(n, n) \in I^+$,*
- *a relation $S \subseteq N \times N$ representing* state transitions *(or* steps*),*
- *a* node labelling ϑ *assigning a finite set of literals $\vartheta(n)$ (over $\mathcal{V}$) to every node $n \in N$.*
- *an* edge labelling ζ *assigning an antecedent $\zeta(n, n')$ (over $\mathcal{V}$) defined as in definition 4.1 to every internal edge $(n, n') \in I$,*
- *and a designated* initial state $n_0 \in N$.

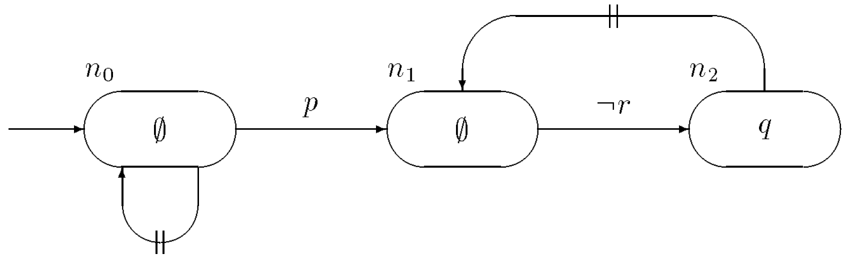


Figure 4.2: Graphical representation of a transition graph

Finite sets $\{L_1, \ldots, L_k\}$ of literals like those appearing as node labels of transition graphs will frequently be identified with the conjunction $L_1 \wedge \ldots \wedge L_k$ of their members where the empty set, as usual, is interpreted as representing t.

Transition graphs can be represented graphically as illustrated by figure 4.2. Nodes and their labels are represented by literal-inscribed ovals, internal edges are represented by arrows whose inscription represents the respective edge label whereas crossed arrows are used for state transitions.

Let us now identify some special forms of transition graphs:

Definition 4.4 (regular transition graphs)
A transition graph $T = (N, I, S, \vartheta, \zeta, n_0)$ is called regular *if for every node $n \in N$ it holds that $I(n) = \emptyset$ or $S(n) = \emptyset$. A node n of a regular transition graph T such that $I(n) = \emptyset$ is called a* state *of T.*

Definition 4.5 (normal transition graphs)
A transition graph $T = (N, I, S, \vartheta, \zeta, n_0)$ is called normal *if for every internal edge $(n, n') \in I$, the edge label $\zeta(n, n')$ is a non-empty finite disjunction $L_1 \vee \ldots \vee L_k$ of literals.*

Definition 4.6 (monotonic transition graphs)
A transition graph $T = (N, I, S, \vartheta, \zeta, n_0)$ is called monotonic *if for all nodes $n, n' \in N$ the following condition holds:*

$$(n, n') \in I \quad \Longrightarrow \quad \vartheta(n) \subseteq \vartheta(n').$$

Definition 4.7 (complete transition graphs)
A transition graph $T = (N, I, S, \vartheta, \zeta, n_0)$ is called complete *if for every node $n \in N$ where $I(n) \neq \emptyset$, the formula $\bigvee_{(n,n') \in I} \zeta(n, n')$ is a (propositional) tautology.*

For example, the transition graph of figure 4.2 is normal and monotonic, but neither regular (the regularity condition is violated by n_0) nor complete (witness n_0 and n_1).

Let us now define what it means for a temporal structure $\mathcal{K}$ (which will again frequently be identified with its state sequence σ) to be accepted by a transition graph. We will introduce markings of transition graphs over a temporal structure in a similar manner as runs of finite automata over finite words are defined. Since transition graphs do not contain final states, the only required acceptance condition will be "absence of failure": The node labels of every node contained in the marking of the transition graph over σ have to be satisfied in the corresponding states of σ (compare this with the more elaborate acceptance condition of Streett automata introduced in definition 4.20 below).

Definition 4.8 (marking over a temporal structure, acceptance)
The marking *of a transition graph $T = (N, I, S, \vartheta, \zeta, n_0)$ over a temporal structure σ is defined as the sequence $M^\sigma = (m_0^\sigma, m_1^\sigma, \ldots)$ where for every $i \geq 0$, $m_i^\sigma \subseteq N$ is the smallest set of nodes satisfying the following conditions:*

- $n_0 \in m_0^\sigma$
- *If $n \in m_i^\sigma$ and $(n, n') \in I$ is an internal edge such that $\sigma_i \models \zeta(n, n')$ then $n' \in m_i^\sigma$.*
- *If $n \in m_i^\sigma$ and $(n, n') \in S$ is a state transition then $n' \in m_{i+1}^\sigma$.*

A temporal structure σ is accepted *by the transition graph T if for the marking M^σ of T over σ the following condition holds:*

$$\sigma_i \models \bigwedge_{n \in m_i^\sigma} \vartheta(n) \quad \text{for every } i \in \mathbb{N}$$

Call a property P (i.e. a set of temporal structures) *definable by transition graphs* if there is a transition graph T accepting exactly the structures in P. Let us now prove that P is definable by transition graphs if and only if $P = Mod(\varphi)$ for some extended program clause φ, i.e. if P is definable by extended program clauses. In fact, effective procedures will be given for the construction of a transition graph accepting exactly the models of a given extended program clause and vice versa.

Algorithm 4.9 (associated transition graph)
Let φ be an extended program clause. Then the transition graph T^φ obtained by the following construction is called the *associated transition graph* for φ.

The algorithm starts by constructing a (generalized) transition graph $\tilde{T} = (N, I, S, \tilde{\vartheta}, \zeta, n_0)$ whose node labelling $\tilde{\vartheta}$ assigns sets of pre-clauses to every node instead of just sets of literals. $T^{\varphi} = (N, I, S, \vartheta, \zeta, n_0)$ is obtained from $\tilde{T}$ by defining the node label $\vartheta(n)$, for every $n \in N$, as the restriction of $\tilde{\vartheta}(n)$ to sets of literals.

Assume, w.l.o.g., that all proposition variables which appear bound in different subformulas $\nu X.F$ of φ are pairwise different, and let, for every X appearing in φ, F_X denote the "defining formula" for X, i.e. the unique formula such that $\nu X.F_X$ appears as a subformula of φ. Since φ by definition has to be a closed formula, such a formula F_X has to exist. Let P be any pre-clause which appears as a subformula of φ and define its *closure set* $cl(P)$ as follows:

$$
\begin{aligned}
& cl(L) = \{L\} \quad \text{if } L \text{ is a literal,} \\
& cl(X) = \{X\} \cup cl(F_X) \quad \text{for a proposition variable } X, \\
& cl(\circ P) = \{\circ P\}, \qquad\qquad cl(P \wedge Q) = \{P \wedge Q\} \cup cl(P) \cup cl(Q), \\
& cl(A \rightarrow P) = \{A \rightarrow P\}, \quad cl(\nu X.F_X) = \{\nu X.F_X\} \cup cl(F_X).
\end{aligned}
$$

For example, for $\varphi \equiv \Box(p \wedge (q \rightarrow \circ r)) \equiv \nu X.p \wedge (q \rightarrow \circ r) \wedge \circ X$, one obtains $cl(\varphi) = \{\varphi, p \wedge (q \rightarrow \circ r) \wedge \circ X, p, q \rightarrow \circ r, \circ X\}$.

Obviously, for every pre-clause P, $cl(P)$ is a finite set of pre-clauses appearing as subformulas of φ and $P \in cl(P)$. Moreover, for every temporal structure σ and every valuation ξ satisfying $\xi(X) = \sigma^{\xi}(\nu X.F_X)$ for every proposition variable X appearing in φ, it holds that $\sigma^{\xi}(P) \subseteq \sigma^{\xi}(F)$ for every $F \in cl(P)$.

Construct $\tilde{T}$ starting from the initial node n_0 which is labelled with $\tilde{\vartheta}(n_0) = cl(\varphi)$, and applying the following rules for the construction of new nodes and edges. Whenever the rule says: "Construct a node n' labelled by $\tilde{\vartheta}(n') = F$ and an edge $(n, n') \in I$ labelled by $\zeta(n, n') = A$" (or $(n, n') \in S$) but some node n'' labelled by F has already been constructed, then only an edge (n, n'') is to be inserted. If such an edge (of the required type) already exists but A does not appear as a disjunct in its label $\zeta(n, n'') = B$ (if it is an internal edge), then the label is changed to $B \vee A$.

1. For every node n whose label $\tilde{\vartheta}(n)$ contains a formula $A \rightarrow P$, construct a node n' labelled by $\tilde{\vartheta}(n') = cl(P)$ and an internal edge $(n, n') \in I$ labelled by $\zeta(n, n') = A$.

2. For every node n whose label $\tilde{\vartheta}(n)$ contains a formula $\circ P$ construct a node n' labelled by $\tilde{\vartheta}(n') = \bigcup\{cl(P) \mid \circ P \in \tilde{\vartheta}(n)\}$ and a state transition $(n, n') \in S$.

This construction introduces only nodes labelled by sets of subformulas of φ of which there is a finite number, hence it will terminate. Notice that the

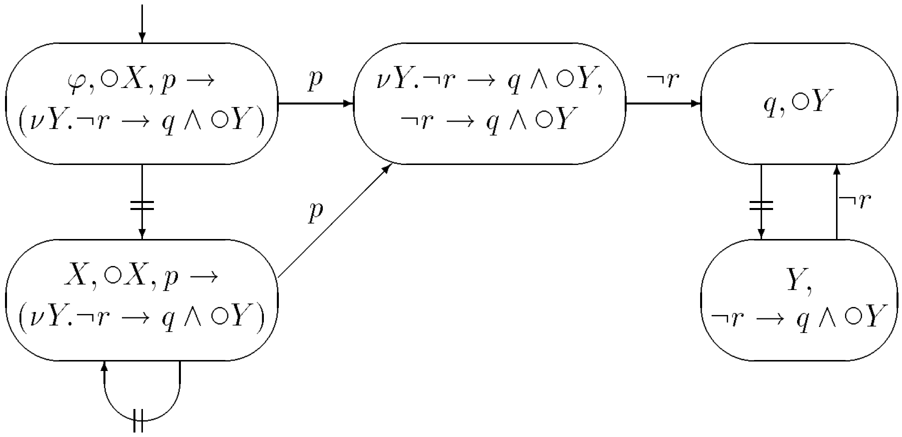


Figure 4.3: A generalized transition graph (example 4.10)

construction ensures that for all pre-clauses P which appear as subformulas of φ and all nodes n, $P \in \tilde{\vartheta}(n)$ implies that $cl(P) \subseteq \tilde{\vartheta}(n)$.

To verify that T^φ is a transition graph, we have to show that the transitive closure I^+ of I is irreflexive. Assume $(n, n') \in I$ for some nodes n, n'. Then $\tilde{\vartheta}(n)$ contains some formula $A \rightarrow P$ such that $\tilde{\vartheta}(n') = cl(P)$. By the construction of $\tilde{T}$, every pre-clause F appearing in some node label $\tilde{\vartheta}(n'')$ where $(n', n'') \in I^*$ either is a subformula of P or of a formula F_Y where the variable Y appears in some label $\tilde{\vartheta}(n''')$ of a node n''' such that $(n', n''') \in I^*$ and $(n''', n'') \in I^*$. In particular, in this case P must contain some variable X such that $\circ$-deg$(X, P) = 0$ (i.e. X is not in the scope of a $\circ$-operator), and if F is not a subformula of P, all occurrences of X in F appear in the scope of at least one $\circ$-operator since this is true for F_X, and $\circ$-formulas are not decomposed in the closure set. Hence, $F \not\equiv A \rightarrow P$, which implies that $n \notin I^+(n)$. ♠

Example 4.10 Figure 4.3 represents the generalized transition graph $\tilde{T}$ which is constructed by algorithm 4.9 for the extended program clause $\varphi \equiv \Box(p \rightarrow q \text{ unless } r) \equiv \nu X.(p \rightarrow (\nu Y.\neg r \rightarrow q \wedge \circ Y)) \wedge \circ X$. The transition graph of figure 4.2 results from $\tilde{T}$ by restricting the node labels to literals and some simplification (similar to the Myhill construction of a minimal finite automaton). ♠

Lemma 4.11 *Let φ be an extended program clause. Then the associated transition graph T^φ accepts exactly the models of φ.*

Proof: Assume $\sigma \models \varphi$ and let $M^\sigma = (m_0^\sigma, m_1^\sigma, \ldots)$ denote the marking of T^φ (notice that the definition of M^σ is independent of the node labelling of a transition graph, hence M^σ is also the marking of the generalized transition graph $\tilde{T}$ constructed in algorithm 4.9 if this notion is defined for generalized transition graphs in the obvious way). Define a partial order $<$ on the proposition variables appearing in φ by $X < Y$ iff $\nu Y.F_Y$ is a subformula of $\nu X.F_X$ (where F_X and F_Y are the "defining" formulas for X and Y as in algorithm 4.9) and define a valuation ξ of the proposition variables by

$$\xi(X) = \sigma^{\xi_X}(\nu X.F_X) \quad \text{where } \xi_X(Y) = \left\{ \begin{array}{ll} \xi(Y) & \text{if } Y < X \\ \emptyset & \text{else} \end{array} \right\}.$$

Notice that by definition of ξ it holds for all variables X appearing in φ that $\xi(X) = \sigma^\xi(F_X)$. Then, for all pre-clauses P appearing as subformulas of φ, $\sigma^\xi(P) = \bigcap\limits_{Q \in cl(P)} \sigma^\xi(Q)$:

- If P is a literal, or if $P \equiv A \to Q$ or $P \equiv \circ Q$, the assertion trivially holds.
- If $P \equiv Q \wedge R$, the assertion follows from the induction hypothesis for Q and R.
- If $P \equiv X$, $\sigma^\xi(X) = \xi(X) = \sigma^\xi(F_X)$ by the definition of $\xi(X)$ as a (greatest) fixpoint valuation.
- For $P \equiv \nu X.F_X$, again by the definition of ξ as a greatest fixpoint valuation, $\sigma^\xi(\nu X.F_X) = \sigma^\xi(F_X)$.

We prove that for every $i \in \mathbb{N}$ and every $n \in m_i^\sigma$,

$$(*) \quad \sigma_i^\xi \models \tilde{\vartheta}(n),$$

where $\tilde{\vartheta}(n)$ is again identified with the conjunction of its members.

In particular, then, $\sigma_i \models \vartheta(n)$ for all $i \in \mathbb{N}$ and all $n \in m_i^\sigma$ since the truth of all formulas in $\vartheta(n)$ is independent of any variable valuation. Hence, T^φ accepts σ.

The proof of $(*)$ runs by induction on the definition of M^σ:

Induction Base: $i = 0, n = n_0$. By the construction of $\tilde{T}$, $\tilde{\vartheta}(n) = cl(\varphi)$, and by the assumption $\sigma \models \varphi$ and because of $\sigma^\xi(\varphi) = \bigcap\limits_{Q \in cl(\varphi)} \sigma^\xi(Q)$, the assertion follows.

Induction Step 1: $n \in m_i^\sigma, (n, n') \in I, \sigma_i \models \zeta(n, n')$. Then n' has been introduced by (possibly multiple) applications of the construction rule 1 of algorithm 4.9, hence there are $k \geq 1$ formulas $A_1 \to P_1, \ldots, A_k \to P_k \in \tilde{\vartheta}(n)$ such that $\tilde{\vartheta}(n') = cl(P_1) = \ldots = cl(P_k)$ and $\zeta(n,n') \equiv A_1 \vee \ldots \vee A_k$ and, for some $1 \leq j \leq k$, $\sigma_i \models A_j$. By the induction hypothesis, $\sigma_i^\xi \models A_j \rightarrow P_j$, yielding $\sigma_i^\xi \models P_j$ and, by the above assertion, $\sigma_i^\xi \models cl(P_j) = \tilde{\vartheta}(n')$.

Induction Step 2: $n \in m_i^\sigma, (n,n') \in S$. The state transition $(n,n') \in S$ has been introduced by rule 2 of algorithm 4.9, hence $\tilde{\vartheta}(n') = \bigcup\{cl(P) \mid \circ P \in \tilde{\vartheta}(n)\}$. Again, by the induction hypothesis, $\sigma_i^\xi \models \tilde{\vartheta}(n)$, in particular, $\sigma_i^\xi \models \bigwedge_{\circ P \in \tilde{\vartheta}(n)} \circ P$, hence $\sigma_{i+1}^\xi \models P$ for every P such that $\circ P \in \tilde{\vartheta}(n)$. By the above assertion, it follows that $\sigma_{i+1}^\xi \models \bigcup_{\circ P \in \tilde{\vartheta}(n)} cl(P) = \tilde{\vartheta}(n')$ as asserted.

This completes the proof of the "if" direction.

Conversely, assume that T^φ accepts σ. Again, consider the marking M^σ of T^φ over σ, and define a valuation ξ of the proposition variables by $\xi(X) = \{i \in I\!N \mid \exists n \in m_i^\sigma \ (X \in \tilde{\vartheta}(n) \text{ or } \nu X.F_X \in \tilde{\vartheta}(n))\}$. We show that for every pre-clause F, for every $i \geq 0$ and every $n \in m_i^\sigma$,

$$(*) \quad \sigma_i^\xi \models F \quad \text{if } F \in \tilde{\vartheta}(n).$$

Since $\varphi \in n_0 \in m_0^\sigma$ by definition of M^σ, it follows in particular that $\sigma_0^\xi \models \varphi$, which is equivalent to $\sigma \models \varphi$ since φ is a closed formula.

The proof of $(*)$ runs by induction on F:

- If F is a literal, then $F \in \tilde{\vartheta}(n)$ implies $F \in \vartheta(n)$ (where ϑ denotes the node labelling of T^φ), hence $\sigma_i \models F$ follows from the acceptance of σ by T^φ.

- If $F \equiv X$ for some proposition variable, the assertion holds by $\sigma^\xi(X) = \xi(X)$ and the definition of ξ.

- If $F \equiv A \rightarrow P$ for an antecedent A and a pre-clause P, then $\tilde{T}$ contains a node n' such that $(n,n') \in I$, $\tilde{\vartheta}(n') = cl(P)$ (hence, in particular, $P \in \tilde{\vartheta}(n')$) and A appears as a disjunct in $\zeta(n,n')$. Thus, if $\sigma_i \models A$, $n' \in m_i^\sigma$ and, by the induction hypothesis, $\sigma_i^\xi \models P$ which proves $\sigma_i^\xi \models A \rightarrow P$.

- If $F \equiv P \wedge Q$ then by the construction of $\tilde{T}$, $P, Q \in \tilde{\vartheta}(n)$, from which $(*)$ follows by induction hypothesis.

- If $F \equiv \circ P$, then there is some n' in $\tilde{T}$ such that $(n,n') \in S$ and $P \in \tilde{\vartheta}(n')$. By the assumption $n \in m_i^\sigma$, it follows that $n' \in m_{i+1}^\sigma$, hence $\sigma_{i+1}^\xi \models P$ by the induction hypothesis, yielding $\sigma_i^\xi \models \circ P$.

- If $F \equiv \nu X.F_X$, assume $j \geq 0$ such that $j \in \xi(X)$. Hence, there is some $n \in m_j^\sigma$ such that $X \in \tilde{\vartheta}(n)$ or $\nu X.F_X \in \tilde{\vartheta}(n)$. In both cases, therefore, $F_X \in \tilde{\vartheta}(n)$ by the construction of $\tilde{T}$. By the induction hypothesis, this implies $\sigma_j^\xi \models F_X$, and since $j \in \xi(X)$ was chosen arbitrarily, $\xi(X) \subseteq \sigma^\xi(F_X)$ and, therefore, $\xi(X) \subseteq \sigma^\xi(\nu X.F_X)$. Now, by the assumptions $\nu X.F_X \in \tilde{\vartheta}(n)$ and $n \in m_i^\sigma$, $i \in \xi(X)$, thus $i \in \sigma^\xi(\nu X.F_X)$ as required. ♠

If φ is a proper program clause, its associated transition graph T^φ has additional properties resulting from the syntactical restrictions on program clauses:

Lemma 4.12 *Let φ be a program clause and T^φ its associated transition graph. Let $\pi = (n^0, n^1, \ldots, n^k)$ be an I^*-path in T^φ, and let $L \in \vartheta(n^k)$ be a literal appearing in the node label of n^k where $L \equiv p$ or $L \equiv \neg p$ for some proposition letter $p \in \mathcal{V}$. Then $q < p$ (w.r.t. the dependency order on literals over $\mathcal{V}$) holds for every proposition letter q which appears in any edge label $\zeta(n^0, n^1), \ldots, \zeta(n^{k-1}, n^k)$ along the path π.*

Proof: We show the following more general assertion concerning the generalized transition graph $\tilde{T}$ of algorithm 4.9: Let $\pi = (n^0, n^1, \ldots, n^k)$ an I^*-path in $\tilde{T}$, $F \in \tilde{\vartheta}(n^k)$ and $p \in \mathit{write}(F)$ for some proposition letter $p \in \mathcal{V}$. Then $q < p$ holds for every proposition letter q appearing in any edge label $\zeta(n^0, n^1), \ldots, \zeta(n^{k-1}, n^k)$ along the path π. By the definition of $\vartheta(n^k)$ as the restriction of $\tilde{\vartheta}(n^k)$ to literals, the statement of the lemma follows for every literal $L \in \vartheta(n^k)$. The proof of the generalized assertion runs by induction on the number k of edges of π: For $k = 0$ there is nothing to prove. Assume the assertion holds for every path consisting of k edges, and consider the path $\pi' = (n^0, n^1, \ldots, n^k, n^{k+1})$. The edge $(n^k, n^{k+1}) \in I$ has been introduced because of, say, $m \geq 1$ pre-clauses $A_1 \rightarrow P_1, \ldots, A_m \rightarrow P_m \in \tilde{\vartheta}(n^k)$, and $\tilde{\vartheta}(n^{k+1}) = \mathit{cl}(P_1) = \ldots = \mathit{cl}(P_m)$. By definition 4.2,

$$\mathit{write}(P_j) = \mathit{write}(A_j \rightarrow P_j) \subseteq \bigcup_{F \in \tilde{\vartheta}(n^k)} \mathit{write}(F),$$

and

$$\mathit{write}(P_j) = \bigcup_{F \in \mathit{cl}(P_j)} \mathit{write}(F) = \bigcup_{F \in \tilde{\vartheta}(n^{k+1})} \mathit{write}(F)$$

for every $1 \leq j \leq m$. By the assumption $p \in \mathit{write}(F)$ for some $F \in \tilde{\vartheta}(n^{k+1})$, it follows that $p \in \mathit{write}(F')$ for some $F' \in \tilde{\vartheta}(n^k)$, hence the induction hypothesis yields that $q < p$ holds for every proposition letter q appearing in the edge labels $\zeta(n^0, n^1), \ldots, \zeta(n^{k-1}, n^k)$ of π. Moreover, since $A_j \rightarrow P_j$ is a sub-pre-clause of the program clause φ, it holds by definition 4.2 that $q < p$ holds for every $q \in \mathit{read}(A_j)$, i.e. for every proposition letter q which appears in the antecedent A_j. The assertion now follows by $\zeta(n^k, n^{k+1}) = A_1 \vee \ldots \vee A_m$. ♠

Moreover, since only module components $p \in \mathcal{V}_\Pi$ are allowed to appear in pre-clauses for proper program clauses, only literals formed from such proposition letters appear in node labels of transition graphs associated with program clauses.

The converse construction, yielding an extended program clause φ^T whose models are exactly the temporal structures accepted by a given transition graph T, is performed by the following algorithm:

Algorithm 4.13 (associated extended program clause)
Let $T = (N, I, S, \vartheta, \zeta, n_0)$ be a transition graph where $N = \{n_0, \ldots, n_k\}$ is the set of nodes of T. The *associated extended program clause* φ^T is inductively defined using the proposition variables $X_0, \ldots, X_k$:

First, define the formulas ψ_j^X and δ_j^X for $0 \le j \le k$ and every subset $X \subseteq \{X_0, \ldots, X_k\}$ of the proposition variables by

$$\begin{aligned} \psi_j^X &\equiv \vartheta(n_j) \wedge \bigwedge_{(n_j,n_l)\in I} (\zeta(n_j, n_l) \to \delta_l^X) \wedge \bigwedge_{(n_j,n_l)\in S} \circ\delta_l^X, \\ \delta_j^X &\equiv \left\{ \begin{array}{ll} X_j & \text{if } X_j \in X \\ \nu X_j.\psi_j^{X \cup \{X_j\}} & \text{otherwise} \end{array} \right\}. \end{aligned}$$

Now, define $\varphi^T :\equiv \delta_0^\emptyset \equiv \nu X_0.\psi_0^{\{X_0\}}$.

By a straightforward induction on $\{X_0, \ldots, X_k\} \setminus X$ it follows that all formulas ψ_i^X and δ_i^X are well-defined pre-clauses whose free proposition variables are among the variables in X. In particular, φ^T is an extended program clause.

Example 4.14 For the transition graph of figure 4.2, algorithm 4.13 yields the extended program clause

$$\nu X_0.(p \to (\nu X_1.\neg r \to \nu X_2.q \wedge \circ X_1)) \wedge \circ X_0$$

which is equivalent to the extended program clause $\Box(p \to q \ \mathsf{unless}\ r)$ of example 4.10 because $\nu X_2.q \wedge \circ X_1$ is equivalent to $q \wedge \circ X_1$ by theorem 2.22(i). ♠

Lemma 4.15 *Let T be a transition graph and φ^T its associated extended program clause. Then, for any temporal structure σ, $\sigma \models \varphi^T$ iff T accepts σ.*

Proof: Assume $\sigma \models \varphi^T$. By induction on the definition of the marking M^σ of T over σ, we show that for every $i \geq 0$ and every node $n_j \in m_i^\sigma$ there is some set $X \subseteq \{X_0, \ldots, X_k\}$ of variables such that $\sigma_i^{\xi_X} \models \delta_j^X$ where ξ_X is a valuation of the proposition variables satisfying $\xi_X(X_l) = \sigma^{\xi_X}(\nu X_l.\psi_l^X)$ for every $X_l \in X$. If $X_j \in X$, this implies $\sigma_i^{\xi_X} \models X_j$, i.e. $i \in \xi_X(X_j) = \sigma^{\xi_X}(\psi_j^X)$ by definition of ξ_X and, in particular, $\sigma_i \models \vartheta(n_j)$ by the definition of ψ_j^X and because the truth of the literals in $\vartheta(n_j)$ in some state is independent of the propositional valuation. If $X_j \notin X$, it follows that $\sigma_i^{\xi_X} \models \nu X_j.\psi_j^{X \cup \{X_j\}}$, which similarly implies $\sigma_i \models \vartheta(n_j)$. Having shown that $\sigma_i \models \vartheta(n_j)$ for every $i \geq 0$ and every $n_j \in m_i^\sigma$, it follows that T accepts σ.

<u>Induction Base:</u> $i = 0, j = 0$. By the assumption, $\sigma_0 \models \varphi^T \equiv \delta_0^\emptyset$, hence the assertion holds for $X = \emptyset$ in which case the valuation ξ_X may be chosen arbitrarily.

<u>Induction Step 1:</u> $n_j \in m_i^\sigma, (n_j, n_l) \in I, \sigma_i \models \zeta(n_j, n_l)$. By the induction hypothesis, $\sigma_i^{\xi_X} \models \delta_j^X$ for some X. If $X_j \in X$, this means $\sigma_i^{\xi_X} \models X_j$, and by $\xi_X(X_j) = \sigma^{\xi_X}(\psi_j^X)$, $\sigma_i^{\xi_X} \models \psi_j^X$. By definition of ψ_j^X and $\sigma_i \models \zeta(n_j, n_l)$, the assertion $\sigma_i^{\xi_X} \models \delta_l^X$ follows.

If $X_j \notin X$, $\sigma_i^{\xi_X} \models \delta_j^X \equiv \nu X_j.\psi_j^{X \cup \{X_j\}}$. Let $J := \sigma^{\xi_X}(\delta_j^X)$ and $\xi' := \xi_X[J/X_j]$. Since X_j does not occur in δ_j^X as a free proposition variable, $\xi'(X_j) = \sigma^{\xi_X}(\delta_j^X) = \sigma^{\xi'}(\delta_j^X) = \sigma^{\xi'}(\nu X_j.\psi_j^{X \cup \{X_j\}})$. For every $X_p \in X$, by definition of ψ_p^X it holds that $\psi_p^X \equiv \psi_p^{X \cup \{X_j\}}[\nu X_j.\psi_j^{X \cup \{X_j\}}/X_j]$, hence it follows that $\xi'(X_p) = \xi_X(X_p) \stackrel{\text{I.H.}}{=} \sigma^{\xi_X}(\nu X_p.\psi_p^X) \stackrel{\text{L.2.15}}{=} \sigma^{\xi_X[J/X_j]}(\nu X_p.\psi_p^{X \cup \{X_j\}}) = \sigma^{\xi'}(\nu X_p.\psi_p^{X \cup \{X_j\}})$, i.e. ξ' may be chosen as the valuation $\xi_{X \cup \{X_j\}}$. Now, using the definition of $\xi'(X_j)$ and $\sigma_i^{\xi_X} \models \delta_j^X$, it follows that $\sigma_i^{\xi'} \models \psi_j^{X \cup \{X_j\}}$ from which one concludes similarly as above that $\sigma_i^{\xi'} \models \delta_l^{X \cup \{X_j\}}$.

<u>Induction Step 2:</u> $n_j \in m_i^\sigma, (n_j, n_l) \in S$. The proof is analogous using the conjunct of ψ_j^X corresponding to the state transition $(n_j, n_l) \in S$.

Conversely, assume σ is accepted by T. Let again $M^\sigma = (m_0^\sigma, m_1^\sigma, \ldots)$ denote the marking of T over σ. We prove, for every set $X \subseteq \{X_0, \ldots, X_k\}$, that

$$(*) \quad \sigma_i^{\xi_X} \models \delta_j^X$$

for every $i \geq 0$, every j such that $n_j \in m_i^\sigma$ and every valuation ξ_X of the proposition variables satisfying $\xi_X(X_l) = \{i \mid n_l \in m_i^\sigma\}$ for every variable $X_l \in X$. In particular, it follows that $\sigma_0^{\xi_\emptyset} \models \delta_0^\emptyset$ for every propositional valuation (since the condition on $\xi_\emptyset$ is trivial), implying the claim $\sigma \models \varphi^T$. The assertion $(*)$ will be proven by induction on $\{X_0, \ldots, X_k\} \setminus X$, the induction hypothesis being that $(*)$ holds for every $\{X_0, \ldots, X_k\} \supseteq X' \supset X$.

Case 1: $X_j \in X$. By $n_j \in m_i^\sigma$ and the definition of ξ_X, it follows that $i \in \xi_X(X_j)$, hence $\sigma_i^{\xi_X} \models X_j \equiv \delta_j^X$.

Case 2: $X_j \notin X$. In this case,

$$\begin{aligned}
\delta_j^X &\equiv \nu X_j.\psi_j^{X \cup \{X_j\}} \\
&\equiv \nu X_j.\vartheta(n_j) \wedge \bigwedge_{(n_j,n_l)\in I} (\zeta(n_j,n_l) \rightarrow \delta_l^{X \cup \{X_j\}}) \wedge \bigwedge_{(n_j,n_l)\in S} \bigcirc\delta_l^{X \cup \{X_j\}}.
\end{aligned}$$

We prove for every $p \geq 0$ such that $n_j \in m_p^\sigma$ and every valuation $\xi_{X \cup \{X_j\}}$ satisfying the above condition for every $X_l \in X \cup \{X_j\}$ that

$$(**) \quad \sigma_p^{\xi_{X \cup \{X_j\}}} \models \psi_j^{X \cup \{X_j\}},$$

yielding

$$\xi_{X \cup \{X_j\}}(X_j) = \{p \geq 0 \mid n_j \in m_p^\sigma\} \subseteq \sigma^{\xi_{X \cup \{X_j\}}}(\psi_j^{X \cup \{X_j\}}),$$

hence also

$$\xi_{X \cup \{X_j\}}(X_j) \subseteq \sigma^{\xi_{X \cup \{X_j\}}}(\nu X_j.\psi_j^{X \cup \{X_j\}}).$$

Together with the assumption $n_j \in m_i^\sigma$, hence $i \in \xi_{X \cup \{X_j\}}(X_j)$, this implies that $\sigma_i^{\xi_{X \cup \{X_j\}}} \models \nu X_j.\psi_j^{X \cup \{X_j\}}$. But the truth of $\nu X_j.\psi_j^{X \cup \{X_j\}}$ in some state does not depend on the valuation of the bound variable X_j, hence also

$$\sigma_i^{\xi_X} \models \nu X_j.\psi_j^{X \cup \{X_j\}} \equiv \delta_j^X$$

which was to be shown. For the proof of $(**)$, assume $p \geq 0$ such that $n_j \in m_p^\sigma$.

1. By the acceptance of σ by T it follows that $\sigma_p \models \vartheta(n_j)$.

2. Let $(n_j, n_l) \in I$ be an internal edge of T such that $\sigma_p \models \zeta(n_j, n_l)$. Then $n_l \in m_p^\sigma$ and by the induction hypothesis (for $(*)$) it follows that $\sigma_p^{\xi_{X \cup \{X_j\}}} \models \delta_l^{X \cup \{X_j\}}$, hence $\sigma_p^{\xi_{X \cup \{X_j\}}} \models \zeta(n_j, n_l) \rightarrow \delta_l^{X \cup \{X_j\}}$ holds for all internal edges $(n_j, n_l) \in I$ of T.

3. If $(n_j, n_l) \in S$ is a state transition of T, then $n_l \in m_{p+1}^\sigma$, and the induction hypothesis for $(*)$ similarly yields $\sigma_{p+1}^{\xi_{X \cup \{X_j\}}} \models \delta_l^{X \cup \{X_j\}}$, i.e. $\sigma_p^{\xi_{X \cup \{X_j\}}} \models \bigcirc\delta_l^{X \cup \{X_j\}}$.

The assertion $(**)$ now follows from the conjunction of 1, 2, and 3 by the definition of $\psi_j^{X \cup \{X_j\}}$. ♠

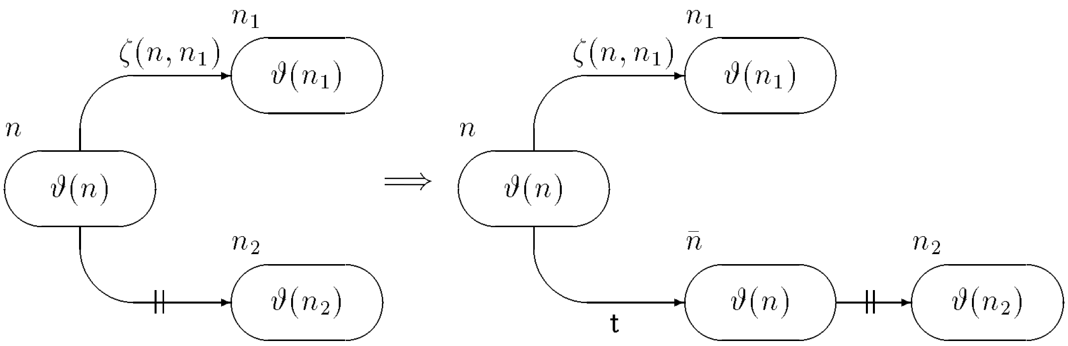


Figure 4.4: Eliminating an irregular node n (lemma 4.16)

The lemmas 4.11 and 4.15 establish that the classes of properties definable by transition graphs and by extended program clauses coincide. This class of properties will be given an independent characterization in the following section. Let us now show that the various restrictions on transition graphs defined above do not change the class of definable properties.

Lemma 4.16 *For every transition graph T there is a regular transition graph T' such that for every temporal structure σ and for every $i \geq 0$,*

$$\bigcup_{n \in m_i} \vartheta(n) = \bigcup_{n' \in m'_i} \vartheta'(n')$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over σ, respectively. In particular, T and T' accept exactly the same temporal structures. Moreover, if T is normal, complete or monotonic, then T' will have the same properties.

Proof: For a transition graph to be regular, there must not exist any node $n \in N$ such that $I(n) \neq \emptyset$ and $S(n) \neq \emptyset$. Assume $T = (N, I, S, \vartheta, \zeta, n_0)$ is a transition graph which contains some node n for which $I(n) \neq \emptyset$ and $S(n) \neq \emptyset$. Construct a transition graph $\bar{T} = (\bar{N}, \bar{I}, \bar{S}, \bar{\vartheta}, \bar{\zeta}, \bar{n}_0)$ by $\bar{N} = N \cup \{\bar{n}\}$ (where $\bar{n} \notin N$ is a new node), $\bar{I}(n') = I(n')$, $\bar{S}(n') = S(n')$, $\bar{\vartheta}(n'') = \vartheta(n'')$ and $\bar{\zeta}(n', n'') = \zeta(n', n'')$ for all nodes $n' \in N \setminus \{n\}$ and $n'' \in N$, $\bar{n}_0 = n_0$. Define $\bar{I}(n) = I(n) \cup \{\bar{n}\}$, $\bar{S}(n) = \emptyset$, $\bar{\zeta}(n, n') = \zeta(n, n')$ if $n' \neq \bar{n}$, $\bar{\zeta}(n, \bar{n}) = \mathsf{t}$, $\bar{I}(\bar{n}) = \emptyset$, $\bar{S}(\bar{n}) = S(n)$ and $\bar{\vartheta}(\bar{n}) = \vartheta(n)$ (see figure 4.4 for a graphical representation of this construction).

Since $\bar{S}(n) = \emptyset$ and $\bar{I}(\bar{n}) = \emptyset$, the number of nodes in $\bar{T}$ which violate the regularity condition has decreased by one, and a straightforward proof establishes that for every temporal structure σ, $\bigcup_{n \in m_i} \vartheta(n) = \bigcup_{n \in \bar{m}_i} \bar{\vartheta}(n)$ where

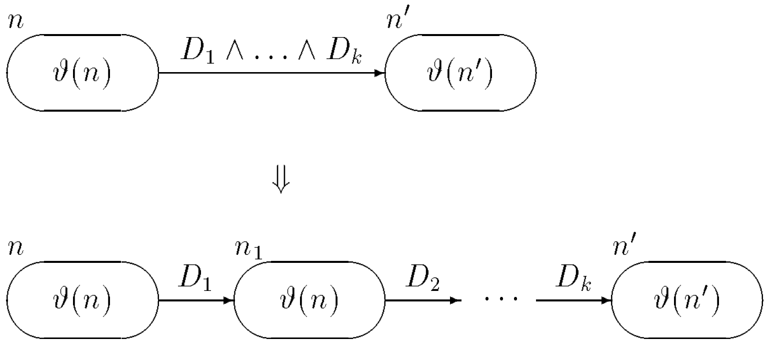


Figure 4.5: Normalizing transition graphs (lemma 4.17)

$M = (m_0, m_1, \ldots)$ and $\bar{M} = (\bar{m}_0, \bar{m}_1, \ldots)$ denote the markings of T and $\bar{T}$ over σ, respectively. Moreover, $\bar{T}$ will be normal, complete or monotonic if T has been so. (If monotonicity were not to be preserved, the definition $\bar{\vartheta}(\bar{n}) = \emptyset$ would also have sufficed.) Applying the same construction a finite number of times therefore yields a regular transition graph T' which satisfies the conditions of the lemma. ♠

Lemma 4.17 *For every transition graph T there is a normal transition graph T' such that for every temporal structure σ and for every $i \geq 0$,*

$$\bigcup_{n \in m_i} \vartheta(n) = \bigcup_{n' \in m'_i} \vartheta'(n')$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over σ, respectively. In particular, T and T' accept exactly the same temporal structures. Moreover, if T is regular or monotonic, then T' will have the same properties.

Proof: Construct the conjunctive normal form of every edge label $\zeta(n, n')$ appearing in T. For every edge label $\zeta(n, n') = D_1 \wedge \ldots \wedge D_k$ (where all D_i are disjunctions of literals) consisting of more than one conjunct, introduce $k - 1$ new nodes $n_1, \ldots, n_{k-1}$ labelled by $\vartheta'(n_i) = \vartheta(n)$ and new internal edges (n, n_1) labelled by $\zeta'(n, n_1) = D_1$, (n_i, n_{i+1}) labelled by $\zeta'(n_i, n_{i+1}) = D_{i+1}$ for $1 \leq i \leq k - 2$ and (n_{k-1}, n') labelled by $\zeta'(n_{k-1}, n') = D_k$ (see figure 4.5).

The resulting transition graph T' will clearly be normal and satisfy the assertion concerning the markings of T and T'. Moreover, the construction preserves regularity and monotonicity. It does not generally preserve completeness. However, the construction of the following lemma 4.18 which generates a complete transition graph without interfering with normality can be applied to T' to regain completeness. ♠

Lemma 4.18 *For every transition graph T there is a complete transition graph T' such that for every temporal structure σ and for every $i \geq 0$,*

$$\bigcup_{n \in m_i} \vartheta(n) = \bigcup_{n' \in m'_i} \vartheta'(n')$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over σ, respectively. In particular, T and T' accept exactly the same temporal structures. Moreover, if T is regular, normal or monotonic, then T' will have the same properties.

Proof: For every node n in T, for which $I(n) \neq \emptyset$ and the disjunction $\bigvee_{(n,n^*) \in I} \zeta(n, n^*)$ is not a tautology, add a new node n' labelled by $\vartheta'(n') = \vartheta(n)$ and an internal edge $(n, n') \in I'$ labelled by $\zeta'(n, n') = \mathsf{t}$. Define $I'(n') = S'(n') = \emptyset$. Obviously, the resulting transition graph is complete and satisfies the assertion of the lemma. Moreover, the construction of T' preserves regularity, normality and monotonicity. ♠

Complete transition graphs T have the pleasant property that for every node $n \in N$ and every valuation $s : \mathcal{V} \to 2^{\{0\}}$ of the proposition letters in $\mathcal{V}$ (in particular, thus, for every state of a temporal structure σ), there is an I^*-path $\pi = (n^0, n^1, \ldots, n^k)$ such that $n^0 = n$, $(n^j, n^{j+1}) \in I$ for $0 \leq j < k$, $I(n^k) = \emptyset$, and all edge labels $\zeta(n^j, n^{j+1})$ for $0 \leq j < k$ are true under the valuation s. Such a path π will be called a *fullpath for n and s*.

Lemma 4.19 *For every transition graph T there is a monotonic transition graph T' such that for every temporal structure σ and for every $i \geq 0$,*

$$\bigcup_{n \in m_i} \vartheta(n) = \bigcup_{n' \in m'_i} \vartheta'(n')$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over σ, respectively. In particular, T and T' accept exactly the same temporal structures. Moreover, if T is regular, normal or complete, then T' will have the same properties.

Proof: The "non-monotonicity" of $T = (N, I, S, \vartheta, \zeta, n_0)$, $nm(T)$ will be measured as follows: For every node n, let $nm(n) = |\{n' \in N \mid (n, n') \in I, \vartheta(n) \not\subseteq \vartheta(n')\}|$ be the number of I-successors of n which violate the monotonicity requirement, and $\ell(n)$ be the length of the longest I^+-path starting at

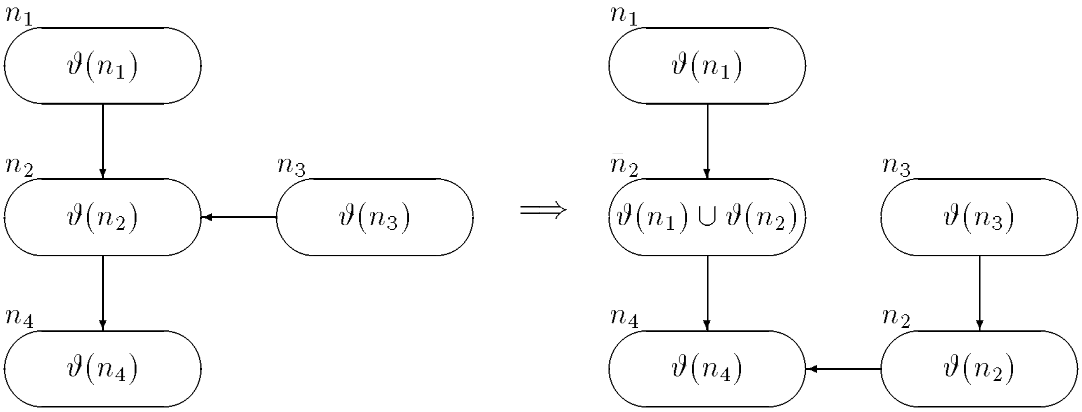


Figure 4.6: Construction step for lemma 4.19

node n, or 0 if n does not have any I-successors (notice that by the irreflexivity of I^+, $\ell(n)$ is well-defined since N is a finite set of nodes). Assume that $k = \max\{\ell(n) \mid n \in N,\ nm(n) \neq 0\}$ is the length of the longest I^+-paths occurring in T which start in a node violating the monotonicity requirement w.r.t. at least one I-successor. Notice that $k = 0$ if and only if T is monotonic. Then $nm(T)$ is defined as the pair $(k, \sum_{\ell(n)=k} nm(n))$. If T is not monotonic, i.e. if $k > 0$, the following construction of a transition graph $\bar{T} = (\bar{N}, \bar{I}, \bar{S}, \bar{\vartheta}, \bar{\zeta}, \bar{n}_0)$ will ensure that $nm(\bar{T}) < nm(T)$ (where $<$ is the lexicographic ordering on pairs of natural numbers) while satisfying $\bigcup_{n \in m_i} \vartheta(n) = \bigcup_{n \in \bar{m}_i(n)} \bar{\vartheta}(n)$ as well as preserving regularity, normality and completeness. Since the lexicographic ordering on $I\!N \times I\!N$ is well-founded, a finite number of iterations will produce a monotonic transition graph T' with the desired properties.

Let $n_1, n_2 \in N$ be nodes such that $(n_1, n_2) \in I$ but $\vartheta(n_1) \not\subseteq \vartheta(n_2)$ and $\ell(n_1) = k$, i.e. n_1 is a "maximal" violator of monotonicity in T and n_2 is one of its non-monotonic I-successors. If for no node $n_3 \in N$, $(n_3, n_2) \in I \cup S$, then $\bar{T} = (N, I, S, \bar{\vartheta}, \zeta, n_0)$ where $\bar{\vartheta}(n) = \vartheta(n)$ for all nodes $n \neq n_2$ and $\bar{\vartheta}(n_2) = \vartheta(n_1) \cup \vartheta(n_2)$. Otherwise, $\bar{N} = N \cup \{\bar{n}_2\}$ (where $\bar{n}_2 \notin N$ is a new node), $\bar{I} = (I \backslash \{(n_1, n_2)\}) \cup \{(n_1, \bar{n}_2)\} \cup \{(\bar{n}_2, n) \mid (n_2, n) \in I\}$, $\bar{S} = S \cup \{(\bar{n}_2, n) \mid (n_2, n) \in S\}$, $\bar{\vartheta}(n) = \vartheta(n)$ for all nodes $n \neq \bar{n}_2$, $\bar{\vartheta}(\bar{n}_2) = \vartheta(n_1) \cup \vartheta(n_2)$, $\bar{\zeta}(n, n') = \zeta(n, n')$ for $(n, n') \in I \setminus \{(n_1, n_2)\}$, $\bar{\zeta}(n_1, \bar{n}_2) = \zeta(n_1, n_2)$, $\bar{\zeta}(\bar{n}_2, n) = \zeta(n_2, n)$ for every $n \in \bar{I}(\bar{n}_2)$, and $\bar{n}_0 = n_0$ (see figure 4.6).

Let $\bar{M}^\sigma$ denote the marking of $\bar{T}$ over the temporal structure σ. For both cases in the above construction step, it is easy to prove that for every temporal structure σ and every $i \in I\!N$, $\bigcup_{n \in m_i^\sigma} \vartheta(n) = \bigcup_{n \in \bar{m}_i^\sigma} \bar{\vartheta}(n)$. On the other hand, $nm(\bar{T}) < nm(T)$ since $\ell(\bar{n}_2) < k$ holds by the choice of n_1, and the number of monotonicity-violating successors of n_1 has decreased without affecting $nm(n)$

for any node $n_1 \neq n \in N \cap \bar{N}$. Clearly, regularity, normality and completeness are preserved by the described construction. ♠

Observe, moreover, that the constructions of the lemmas 4.16–4.18 preserve the condition of lemma 4.12 concerning the dependency order of literals appearing in node and edge labels of transition graphs corresponding to proper program clauses. The construction of lemma 4.19 producing a monotonic transition graph preserves the weaker condition that for every I^+-path $\pi = (n^0, \ldots, n^k)$, $q < p$ holds for every proposition letter q appearing in one of the edge labels $\zeta(n^0, n^1), \ldots, \zeta(n^{k-1}, n^k)$ and every proposition letter p such that $(\neg)p \in \vartheta(n^k) \setminus \vartheta(n^{k-1})$.

4.3 Extended program clauses express deterministic safety properties

It has been shown in the preceding section that extended program clauses and transition graphs agree in the class of properties (i.e. sets of temporal structures) they can define. It is interesting to note that this class of properties coincides with a subclass of safety properties which has previously been studied in the literature and which is defined independently of temporal logic by certain finite automata. This digression from our study of the imperative view of νTL formulas will be pursued in this section.

For the definition of properties by means of automata we follow Manna & Pnueli [37], but see also Alpern & Schneider [8] or Vardi [56].

Definition 4.20 (Streett automata)

(i) *A finite* Streett automaton $\mathcal{A} = (Q, q_0, T, L)$ *consists of:*

- *a finite set Q of* (automaton) states*,*
- *a designated* initial state $q_0 \in Q$ *of $\mathcal{A}$,*
- *a set $T = \{t(q, q') \mid q, q' \in Q\}$ of* transition conditions *(expressed as propositional formulas over $\mathcal{V}$) where $t(q, q')$ gives the condition for a transition from automaton state q to automaton state q',*
- *a finite set $L = \{(R_1, P_1), \ldots (R_n, P_n)\}$ of pairs of sets $R_i, P_i \subseteq Q$. (The states in R_i are called* recurrent states*, the states in P_i are called* persistent states*.)*

(ii) *Let $\sigma = (s_0, s_1, \ldots)$ be a temporal structure and let $\mathcal{A} = (Q, q_0, T, L)$ be a Streett automaton. An ω-sequence $\rho = (r_0, r_1, \ldots)$ of automaton states $r_i \in Q$ is called a* run of $\mathcal{A}$ over σ *if $r_0 = q_0$ is the initial state of $\mathcal{A}$ and if for every $i \geq 0$, $\sigma_i \models t(r_i, r_{i+1})$. For a run ρ of $\mathcal{A}$ over σ, the* infinity set *of ρ, denoted by $\mathrm{Inf}(\rho) \subseteq Q$, is the set of automaton states which appear infinitely often in ρ.*

(iii) *A run ρ of $\mathcal{A}$ over a temporal structure σ is called* accepting *if, for each $i = 1, \ldots, n$, $\mathrm{Inf}(\rho) \cap R_i \neq \emptyset$ or $\mathrm{Inf}(\rho) \subseteq P_i$. $\mathcal{A}$* accepts *σ if there is some accepting run ρ of $\mathcal{A}$ over σ.*

(iv) *A Streett automaton $\mathcal{A} = (Q, q_0, T, L)$ is called* complete *if for all $q \in Q$ the disjunction $\bigvee_{q' \in Q} t(q, q')$ is a (propositional) tautology.*

(v) *A Streett automaton $\mathcal{A} = (Q, q_0, T, L)$ is called* deterministic *if for every q and all $q' \neq q''$, the formula $t(q, q') \rightarrow \neg t(q, q'')$ is a tautology (i.e. for every automaton state q at most one transition condition $t(q, q')$ can be true in any state of a temporal structure).*

(vi) *A* safety automaton *is a complete and deterministic Streett automaton $\mathcal{A} = (Q, q_0, T, L)$ where L contains only one pair (R, P) of sets and, moreover, for all $q \in R \cup P$, $q' \in Q \setminus (R \cup P)$, the transition condition $t(q, q')$ is a contradiction, i.e. $\neg t(q, q')$ is a tautology. A safety automaton will often be denoted by $\mathcal{A} = (Q, q_0, T, R, P)$; R (resp., P) is called the set of recurrent (resp., persistent) states of $\mathcal{A}$.*

Observe that for every complete and deterministic Streett automaton (hence, in particular, for every safety automaton) $\mathcal{A}$ and every temporal structure σ there is exactly one run of $\mathcal{A}$ over σ. The intuition behind the definition of safety automata is as following: Let $G := R \cup P$ denote the set of "good" states, and $B := Q \setminus G$ the set of "bad" states. Then a safety automaton can never make a transition leading from a "bad" state to a "good" one, hence it cannot accept any computation sequence which causes it to enter a "bad" state once. Consequently, the original acceptance condition for Streett automata is unnecessarily complicated for safety automata. Rather, the temporal structure σ is accepted by a safety automaton $\mathcal{A}$ iff for the run $\rho = (r_0, r_1, \ldots)$ of $\mathcal{A}$ over σ it holds that $r_i \in G$ for every $i \geq 0$. In fact, the distinction between recurrent and persistent states becomes unnecessary for safety automata.

Let Σ, as in section 2.6, be the set of all possible states, i.e. of all mappings: $\mathcal{V} \rightarrow 2^{\{0\}}$. A property $P \subseteq \Sigma^\omega$ is *defined by a Streett automaton $\mathcal{A}$* if $\mathcal{A}$ accepts exactly the temporal structures $\sigma \in P$. It is a well-known result (cf., e.g., Thomas [54] and Stirling [47]) that a property $P \subseteq \Sigma^\omega$ is expressible as $Mod(F)$ for some νTL formula F if and only if it is definable by some Streett automaton $\mathcal{A}$. In the sequel we will study a more restricted problem of this kind, viz. the equivalence between definability by safety automata and by extended program clauses.

In contrast to finite automata operating on finite words, where there is no difference between deterministic and non-deterministic automata in terms of the classes of definable languages, it does not hold that for every non-deterministic Streett automaton $\mathcal{A}$ there is a deterministic Streett automaton $\mathcal{A}'$ such that both automata define the same property. Let us call a property P *deterministic* if it can be defined by a deterministic Streett automaton. The following fact has been proven in MANNA & PNUELI [37] using earlier results by LANDWEBER [32] and WAGNER [57]:

Fact 4.21 *A property is a deterministic safety property iff it is definable by a safety automaton.*

We will prove in the sequel that the class of deterministic safety properties, moreover, coincides with the class of models of extended program clauses.

Lemma 4.22 *For every safety automaton $\mathcal{A}$ there is an extended program clause $\varphi^{\mathcal{A}}$ such that for all state sequences σ the following equivalence holds:*

$$\mathcal{A} \textit{ accepts } \sigma \quad \Longleftrightarrow \quad \sigma \models \varphi^{\mathcal{A}}$$

Proof: If the initial state q_0 of $\mathcal{A} = (Q, q_0, T, R, P)$ is not contained in the set $G := P \cup R$ of "good" states, then $\mathcal{A}$ cannot accept any temporal structure σ, and choosing $\varphi^{\mathcal{A}} := \mathsf{f}$ validates the assertion.

Otherwise, assume $Q = \{q_0, \ldots, q_n\}$ and $G = \{q_0, \ldots, q_k\}$ for some $k \leq n$. Moreover, w.l.o.g., assume that all transition conditions $t(q_i, q_j)$ are formulas in positive normal form, i.e. they are antecedents as defined in definition 4.1. The extended program clause $\varphi^{\mathcal{A}}$ is built from $\mathcal{V}$ and the proposition variables $X_0, \ldots, X_k$. Define, for every set $X \subseteq \{X_0, \ldots, X_k\}$ and $0 \leq j \leq k$, the formulas ψ_j^X and δ_j^X by the following inductive definitions:

$$\begin{aligned}
\psi_j^X &\equiv \bigwedge_{l=0}^{k}(t(q_j, q_l) \rightarrow \bigcirc\delta_l^X) \ \wedge \bigwedge_{l=k+1}^{n}(t(q_j, q_l) \rightarrow \mathsf{f}), \quad \text{and} \\
\delta_j^X &\equiv \left\{ \begin{array}{ll} X_j & \text{if } X_j \in X \\ \nu X_j . \psi_j^{X \cup \{X_j\}} & \text{otherwise} \end{array} \right\}.
\end{aligned}$$

Now, let $\varphi^{\mathcal{A}} :\equiv \delta_0^{\emptyset} \equiv \nu X_0 . \psi_0^{\{X_0\}}$. A simple induction on $\{X_0, \ldots, X_k\} \setminus X$ shows that all formulas ψ_j^X and δ_j^X are well-defined pre-clauses (notice that this is the reason for writing $t(q_j, q_l) \rightarrow \mathsf{f}$ instead of the logically equivalent formula $\neg t(q_j, q_l)$ which might not be a pre-clause) whose free proposition variables are contained in X. In particular, $\varphi^{\mathcal{A}}$ is an extended program clause. We have to prove that for all $\sigma \in \Sigma^\omega$, $\mathcal{A}$ accepts σ iff $\sigma \models \varphi^{\mathcal{A}}$.

$\Longrightarrow$: Since $\mathcal{A}$ is a complete and deterministic automaton, to every state sequence σ there corresponds a unique run $\rho = (r_0, r_1, \ldots)$ where $r_i \in Q$ for all $i \geq 0$, $r_0 = q_0$, and $\sigma_i \models t(r_i, r_{i+1})$. By the assumption that σ is accepted by $\mathcal{A}$, it follows that $r_i \in G$ for all $i \geq 0$ because $\mathcal{A}$ is a safety automaton. Let for $X \subseteq \{X_0, \ldots, X_k\}$, ξ_X denote any valuation of the proposition variables satisfying $\xi_X(X_j) = \{i \mid r_i = q_j\}$ for every $X_j \in X$. We prove that for every $X \subseteq \{X_0, \ldots, X_k\}$, every $i \geq 0$ and $0 \leq j \leq k$,

$$(*) \quad \sigma_i^{\xi_X} \models \delta_j^X \quad \text{if } r_i = q_j$$

In particular, $(*)$ implies that $\sigma_0^{\xi_\emptyset} \models \delta_0^\emptyset \equiv \varphi^{\mathcal{A}}$ (since $r_0 = q_0$) and therefore, $\sigma \models \varphi^{\mathcal{A}}$.

The proof of $(*)$ runs by induction on $\{X_0, \ldots, X_k\} \setminus X$. Assume that $(*)$ holds for all $X' \supset X$ and let $r_i = q_j$. Now, there are two cases:

Case 1: $X_j \in X$. Then $\delta_j^X \equiv X_j$ and by $r_i = q_j$ and the definition of ξ_X it follows that $i \in \xi_X(X_j)$, hence $\sigma_i^{\xi_X} \models \delta_j^X$ as required.

Case 2: $X_j \notin X$. Then

$$\delta_j^X \equiv \nu X_j.\psi_j^{X \cup \{X_j\}} \equiv \nu X_j. \bigwedge_{l=0}^{k} (t(q_j, q_l) \rightarrow \circ\delta_l^{X \cup \{X_j\}}) \;\wedge \bigwedge_{l=k+1}^{n} (t(q_j, q_l) \rightarrow \mathsf{f}).$$

We show that for every $p \geq 0$ such that $r_p = q_j$,

$$(**) \quad \sigma_p^{\xi_{X \cup \{X_j\}}} \models \psi_j^{X \cup \{X_j\}}.$$

This implies

$$\xi_{X \cup \{X_j\}}(X_j) = \{P \mid r_p = q_j\} \subseteq \sigma^{\xi_{X \cup \{X_j\}}}(\psi_j^{X \cup \{X_j\}}),$$

hence also

$$\xi_{X \cup \{X_j\}}(X_j) \subseteq \sigma^{\xi_{X \cup \{X_j\}}}(\nu X_j.\psi_j^{X \cup \{X_j\}}).$$

By the assumption of $(*)$ that $r_i = q_j$, hence $i \in \xi_{X \cup \{X_j\}}(X_j)$, and the independence of the truth of $\nu X_j.\psi_j^{X \cup \{X_j\}}$ of the valuation of the bound proposition variable X_j, it follows that

$$\sigma_i^{\xi_X} \models \nu X_j.\psi_j^{X \cup \{X_j\}} \equiv \delta_j^X,$$

establishing $(*)$. For the proof of $(**)$, assume $p \geq 0$ such that $r_p = q_j$. Since $r_{p+1} \in G$ (see above), say $r_{p+1} = q_l$ where $0 \leq l \leq k$, by the definition of the run of $\mathcal{A}$ over σ it follows that $\sigma_p \models t(q_j, q_l)$. Since $\mathcal{A}$ is deterministic, this implies $\sigma_p \models \bigwedge_{m \neq l} \neg t(q_j, q_m)$, and by the induction hypothesis (for $(*)$), $\sigma_{p+1}^{\xi_{X \cup \{X_j\}}} \models \delta_l^{X \cup \{X_j\}}$. Together we obtain $\sigma_p^{\xi_{X \cup \{X_j\}}} \models \psi_j^{X \cup \{X_j\}}$.

$\Longleftarrow$: Assume $\sigma \models \varphi^{\mathcal{A}}$. Let again $\rho = (r_0, r_1, \ldots)$ denote the (unique) run of $\mathcal{A}$ over σ. We will show that for every $i \geq 0$ there is some $0 \leq j \leq k$ such that $r_i = q_j$ and some $X \subseteq \{X_0, \ldots, X_k\}$ such that $\sigma_i^{\xi_X} \models \delta_j^X$ where $\xi_X : X \to I\!N$ is a propositional valuation satisfying $\xi_X(X_l) = \sigma^{\xi_X}(\nu X_l.\psi_l^X)$ for every proposition variable $X_l \in X$. In particular, $r_i \in G$ for every $i \geq 0$, i.e. $\mathcal{A}$ accepts σ. The proof runs by induction on i:

Induction Base ($i = 0$): By the assumption $\sigma \models \varphi^{\mathcal{A}} \equiv \delta_0^{\emptyset}$, the assertion holds for $X = \emptyset$, $j = 0$ (since $r_0 = q_0$) and an arbitrary valuation $\xi_\emptyset$.

Induction step: By the induction hypothesis, $r_i = q_j$ for some $0 \leq j \leq k$, and there is some $X \subseteq \{X_0, \ldots, X_k\}$ such that $\sigma_i^{\xi_X} \models \delta_j^X$. By the condition on the valuation ξ_X, it follows similarly as in lemma 4.15 that $\sigma_i^{\xi_{X \cup \{X_j\}}} \models \psi_j^{X \cup \{X_j\}}$. In particular, $\sigma_i^{\xi_{X \cup \{X_j\}}} \models \bigwedge_{l=k+1}^{n} \neg t(q_j, q_l)$. Since $\mathcal{A}$ is a complete Streett automaton, $\sigma_i \models \bigvee_{l=0}^{n} t(q_j, q_l)$, hence there is some $0 \leq l \leq k$ such that $\sigma_i \models t(q_j, q_l)$, and $r_{i+1} = q_l$ because $\mathcal{A}$ is deterministic. But then, $\sigma_i^{\xi_{X \cup \{X_j\}}} \models \psi_j^{X \cup \{X_j\}}$ implies $\sigma_i^{\xi_{X \cup \{X_j\}}} \models \circ\delta_l^{X \cup \{X_j\}}$ (by the definition of $\psi_j^{X \cup \{X_j\}}$), i.e. $\sigma_{i+1}^{\xi_{X \cup \{X_j\}}} \models \delta_l^{X \cup \{X_j\}}$ which was to be shown. ♠

To prove the converse assertion, i.e. that every property definable by some extended program clause can also be defined by some safety automaton, we make use of the equivalence between extended program clauses and transition graphs established in the previous section.

Lemma 4.23 *For every transition graph G there is a safety automaton $\mathcal{A}$ which accepts exactly the temporal structures accepted by G.*

Proof: $G = (N, I, S, \vartheta, \zeta, n_0)$ will be assumed to be regular, complete, and monotonic. This restriction can w.l.o.g. be imposed by the lemmas 4.16, 4.18 and 4.19. Let $\mathcal{V}$ be the (finite) set of proposition letters appearing in the node and edge labels of G. Let $N^+ \subseteq N$ denote the set of nodes $N^+ := \{n_0\} \cup \{n \in N \mid \exists n' \in N\ (n', n) \in S\}$. Define the safety automaton $\mathcal{A} = (Q, q_0, T, R, P)$ by $Q := 2^{N^+} \cup \{q_f\}$ where q_f is a new node (taken to represent failure), $q_0 := \{n_0\}$, $R := P := Q \setminus \{q_f\}$. The set T of transition conditions is defined

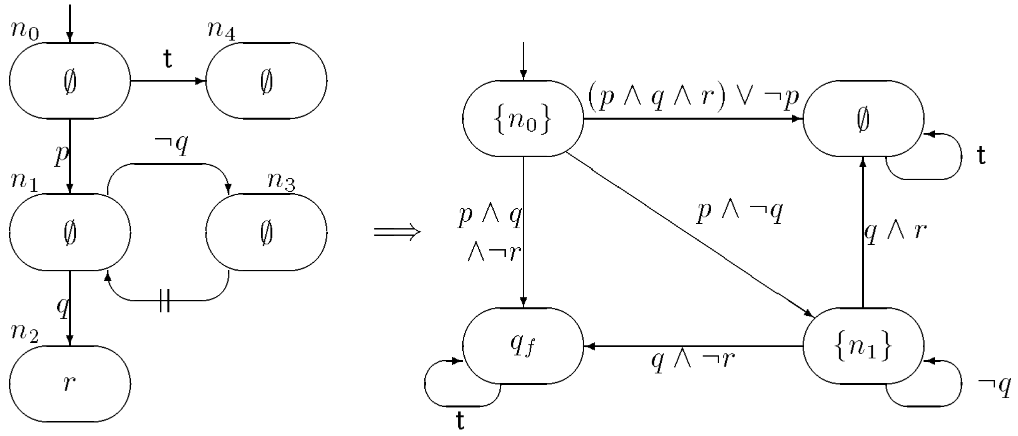


Figure 4.7: Constructing a safety automaton from a transition graph

as follows: Let $q_f \neq q = \{n_1, \ldots, n_k\}$ (where $k \geq 0$) be some node of $\mathcal{A}$. Consider the (finite) set of valuations $\xi : \mathcal{V} \to 2^{\{()\}}$ of the proposition letters in $\mathcal{V}$ and, for every valuation ξ, its *characteristic formula* $F(\xi)$ defined by

$$F(\xi) = \bigwedge_{\xi(v)=\{()\}} v \ \wedge \bigwedge_{\xi(v)=\emptyset} \neg v.$$

(Notice that $F(\xi) \to \neg F(\xi')$ is a tautology for any two different valuations ξ and ξ'.)

Every valuation ξ defines the set $\{\pi_1^\xi, \ldots, \pi_m^\xi\}$ of fullpaths for the nodes $n_j \in q$ and ξ which end in states $e_1^\xi, \ldots, e_m^\xi$ of G, i.e. nodes satisfying $I(e_i^\xi) = \emptyset$. Call ξ an *error valuation for* q if there is at least one path π_i^ξ ending in a state e_i^ξ whose node label $\vartheta(e_i^\xi)$ is not true under the valuation ξ. In this case, define the "successor of q under ξ", $s(q, \xi) := q_f$. Otherwise, let $s(q, \xi) = \bigcup_{i=1}^m S(e_i^\xi)$ be the set of S-successors of the states e_i^ξ. Now, define the transition conditions $t(q, q')$ by

$$\begin{aligned} t(q, q') &= \bigvee_{\{\xi \mid s(q,\xi)=q'\}} F(\xi) \quad \text{for every } q \neq q_f, \\ t(q_f, q_f) &= \mathsf{t}, \\ t(q_f, q') &= \mathsf{f} \quad \text{for every } q' \neq q_f \end{aligned}$$

(where the empty disjunction is, as usually, defined as f). Observe, in particular, that this definition implies $t(\emptyset, \emptyset) = \mathsf{t}$ and $t(\emptyset, q') = \mathsf{f}$ for every $q' \neq \emptyset$.

Figure 4.7 shows an example of the construction of a safety automaton $\mathcal{A}$ corresponding to a regular, complete and monotonic transition graph G (which in turn corresponds to the extended program clause $p \to r$ atnext q). Only the reachable automaton states of $\mathcal{A}$ are displayed. As an example, the constructions leading to the transition conditions for edges leaving node $\{n_0\}$ of $\mathcal{A}$ are given in the following table (the actual edge inscriptions displayed in figure 4.7 result from the above definitions by propositional simplifications):

F_ξ	e_i^ξ	ξ error val.	$s(\{n_0\},\xi)$
$\neg p \wedge \neg q \wedge \neg r$	n_4	no	$\emptyset$
$\neg p \wedge \neg q \wedge r$	n_4	no	$\emptyset$
$\neg p \wedge q \wedge \neg r$	n_4	no	$\emptyset$
$\neg p \wedge q \wedge r$	n_4	no	$\emptyset$
$p \wedge \neg q \wedge \neg r$	n_3	no	n_1
$p \wedge \neg q \wedge r$	n_3	no	n_1
$p \wedge q \wedge \neg r$	n_2	yes	q_f
$p \wedge q \wedge r$	n_2	no	$\emptyset$

The Streett automaton $\mathcal{A}$ resulting from the given transition graph G by the described construction is certainly complete and deterministic: for every $q \in Q$ and every valuation ξ of the variables appearing in the edge labels of $\mathcal{A}$, $F(\xi)$ appears as a disjunct in one of the transition conditions $t(q, q')$. For every $q \in Q$ and all $q' \neq q'' \in Q$, the transition conditions $t(q, q')$ and $t(q, q'')$ are exclusive by the above observation that $F(\xi) \to \neg F(\xi')$ for $\xi \neq \xi'$. Moreover, $R \cup P = Q \setminus \{q_f\}$, and $T(q_f, q') = \mathsf{f}$ for $q' \neq q_f$, hence $\mathcal{A}$ is a safety automaton. It remains to show that G and $\mathcal{A}$ accept the same temporal structures σ. Consider the marking $M^\sigma = (m_0^\sigma, m_1^\sigma, \ldots)$ of G over σ and the (unique) run $\rho = (r_0, r_1, \ldots)$ of $\mathcal{A}$ over σ. We prove that for every $i \geq 0$ the following condition holds:

$$(*) \quad r_{i+1} \neq q_f \iff r_{i+1} = \bigcup_{n \in m_i^\sigma} S(n) \iff \sigma_j \models \bigcup_{n \in m_j^\sigma} \vartheta(n) \text{ for all } j \leq i.$$

(Recall that sets of literals are identified with the conjunction of their elements.)

Now, $\mathcal{A}$ accepts σ iff $r_i \neq q_f$ for every $i > 0$ iff (by $(*)$) $\sigma_i \models \bigcup_{n \in m_i^\sigma} \vartheta(n)$ for every $i \geq 0$ iff G accepts σ. The proof of $(*)$ proceeds by induction on $i \geq 0$:

<u>Induction Base:</u> $i = 0$. By definition, $r_0 = \{n_0\}$. Let $\{e_1^{\sigma[0]}, \ldots, e_m^{\sigma[0]}\}$ be the end nodes of all fullpaths for n_0 and the propositional valuation $\sigma[0]$. Since G is monotonic, it holds that $\bigcup_{n \in m_0^\sigma} \vartheta(n) = \bigcup_{j=1}^{m} \vartheta(e_j^{\sigma[0]})$, and from the regularity of G it follows that $\bigcup_{n \in m_0^\sigma} S(n) = \bigcup_{j=1}^{m} S(e_j^{\sigma[0]})$. Now, either $\sigma_0 \models \bigcup_{j=1}^{m} \vartheta(e_j^{\sigma[0]})$, implying

$\sigma_0 \models \bigcup\limits_{n \in m_0^\sigma} \vartheta(n)$ and $r_1 = s(\{n_0\}, \sigma[0]) = \bigcup\limits_{j=1}^{m} S(e_j^{\sigma[0]}) = \bigcup\limits_{n \in m_0^\sigma} S(n) \neq q_f$, or else $\sigma_0 \not\models \bigcup\limits_{n \in m_0^\sigma} \vartheta(n)$ and $r_1 = q_f \neq \bigcup\limits_{n \in m_0^\sigma} S(n)$. In both cases, the assertion $(*)$ holds.

Induction Step: $i > 0$. If $r_i = q_f$, then by the induction hypothesis there is some $j \leq i-1$ such that $\sigma_j \not\models \bigcup\limits_{n \in m_j^\sigma} \vartheta(n)$, and by $t(q_f, q_f) = \mathsf{t}$, $r_{i+1} = q_f \neq \bigcup\limits_{n \in m_i \sigma} S(n)$.

If $r_i \neq q_f$, then by the induction hypothesis $r_i = \bigcup\limits_{n \in m_{i-1}^\sigma} S(n)$ and $\sigma_j \models \bigcup\limits_{n \in m_j^\sigma} \vartheta(n)$ for every $j \leq i-1$. Applying exactly the same arguments as in the case $i = 0$ above (using the end nodes $e_j^{\sigma[i]}$ of fullpaths for the nodes $n \in r_i$ and the valuation $\sigma[i]$ instead of the fullpaths for n_0 and $\sigma[0]$), it follows that $r_{i+1} = \bigcup\limits_{n \in m_i^\sigma} S(n)$ and $\sigma_i \models \bigcup\limits_{n \in m_i^\sigma} \vartheta(n)$.

In both cases, the assertion $(*)$ is again established. ♠

Combining the results of fact 4.21, lemmas 4.22 and 4.23, as well as 4.11 and 4.15, it follows that extended program clauses, transition graphs and safety automata all define the same class of properties.

Theorem 4.24 *For every property $P \subseteq \Sigma^\omega$ (where Σ is the set of mappings $\mathcal{V} \to 2^{\{()\}}$ for a given set $\mathcal{V}$ of proposition letters), the following assertions are equivalent:*

(i) *There is an extended program clause φ such that $Mod(\varphi) = P$ (P is definable by extended program clauses).*

(ii) *There is a transition graph G accepting exactly the structures in P (P is definable by transition graphs).*

(iii) *There is a safety automaton $\mathcal{A}$ accepting exactly the structures in P (P is definable by safety automata).*

(iv) *P is a deterministic safety property.*

Theorem 4.24 leaves us with two technically different automata-theoretic representations of extended program clauses. For every single state transition of the safety automaton representing a clause, the corresponding transition graph makes several internal transitions, followed by one state transition for every fullpath corresponding to the internal transitions. Thus, the transition graph representation of extended program clauses is more fine-grained than the representation by safety automata. This makes transition graphs better suited for the construction of models of (proper) program clauses. Therefore, only this representation will be used in the sequel. Safety automata, on the other hand, are better suited for model checking (i.e. testing the validity of some formula in a given structure).

4.4 Conflict graphs

Let us now come back to the definition of an operational semantics for (proper) program clauses. Transition graphs will be used to represent program clauses, and the "execution" of program clauses will consist in calculating, incrementally, the markings of these transition graphs. The restrictions placed on the syntax of program clauses (versus extended program clauses) will ensure that this calculation can be accomplished by a straightforward, backtracking-free algorithm of linear complexity (measured in the combined size of all transition graphs). The literals appearing in node labels of nodes representing one state of the temporal structure will be used to define this state. Before presenting the interpreter algorithm in detail in the next section, however, one problem will be addressed now: How can it be ensured that the union of all labels of nodes representing one state is a satisfiable set of literals? If this condition were not satisfied, there would be two (not necessarily different) nodes and some state component $p \in \mathcal{V}_\Pi$ such that one node contains p and the other one $\neg p$ in their node labels. (Observe that $\mathsf{f} \in \vartheta(n)$ where n is a node of a transition graph corresponding to a proper program clause is impossible because node labels of such transition graphs only contain literals formed from module state components $p \in \mathcal{V}_\Pi$.) Clearly, the model construction would not succeed and the νTL interpreter would report an error, similar to the run-time error generated by a conventional program written in a programming language allowing multiple assignments (such as the UNITY programming language used in Chandy & Misra [14]) when a statement like

$$x, x := e_1, e_2$$

is encountered, and the values denoted by the expressions e_1 and e_2 are different. (It may be argued, however, that such a situation can be avoided much more easily in conventional programming languages than in νTL programs.) Chandy and Misra write about such situations ([14, p. 24]):

> *A variable may appear more than once in the left side of an assignment-statement. It is the programmer's responsibility to en-*

sure for any such variable that all possible values that may be assigned to it in a statement are identical.

Similarly, the developer of a νTL program will have to ensure that all possible markings of the transition graphs representing the program yield satisfiable sets of node labels. Fortunately, the representation of program clauses by transition graphs helps in the precise statement of proof obligations guaranteeing this consistency requirement. The necessary technical apparatus is provided by the conflict graphs to be introduced in this section.

Algorithm 4.25 (conflict graph) *Let $T^1 = (N^1, I^1, S^1, \vartheta^1, \zeta^1, n_0^1)$ and $T^2 = (N^2, I^2, S^2, \vartheta^2, \zeta^2, n_0^2)$ be two regular and normal transition graphs. The* conflict graph *$T = (N, I, S, \vartheta, \zeta, n_0)$ of T^1 and T^2 is the transition graph where N, n_0, I, S and ζ are defined as follows:*

- $N \subseteq N^1 \times N^2$ is the smallest set of nodes satisfying the following conditions.
- $n_0 = (n_0^1, n_0^2) \in N$.
- If $(n^1, n^2) \in N$ is a node of T and $I^1(n^1) \neq \emptyset$, then $I(n^1, n^2) = I^1(n^1) \times \{n^2\}$ (in particular, $I^1(n^1) \times \{n^2\} \subseteq N$), $S(n^1, n^2) = \emptyset$, and $\zeta((n^1, n^2), (n, n^2)) = \zeta^1(n^1, n)$ for every $(n, n^2) \in I(n^1, n^2)$.
- If $(n^1, n^2) \in N$ is a node of T and $I^1(n^1) = \emptyset$ but $I^2(n^2) \neq \emptyset$, then $I(n^1, n^2) = \{n^1\} \times I^2(n^2)$ (in particular, $\{n^1\} \times I^2(n^2) \subseteq N$), $S(n^1, n^2) = \emptyset$, and $\zeta((n^1, n^2), (n^1, n)) = \zeta^2(n^2, n)$ for every $(n^1, n) \in I(n^1, n^2)$.
- If $(n^1, n^2) \in N$ is a node of T and $I^1(n^1) = I^2(n^2) = \emptyset$, then $I(n^1, n^2) = \emptyset$ and $S(n^1, n^2) = S^1(n^1) \times S^2(n^2)$ (in particular, $S^1(n^1) \times S^2(n^2) \subseteq N$).

The node labelling ϑ of T is given by the following algorithm:

Initialization: For every $(n^1, n^2) \in N$, set $\vartheta(n^1, n^2) := \emptyset$.

Iteration: While there exists some (unmarked, see below) node $(n^1, n^2) \in N$ such that $\vartheta^1(n^1) \cup \vartheta^2(n^2) \cup \vartheta(n^1, n^2)$ contains either f or a complementary pair $p, \neg p$ of literals (for some proposition letter $p \in \mathcal{V}$):

- Set $\vartheta(n^1, n^2) := \vartheta(n^1, n^2) \cup \{\mathsf{f}\}$.

- For every $(n', n'') \in N$ such that $((n', n''), (n^1, n^2)) \in I$, assuming $\zeta((n', n''), (n^1, n^2)) = L_1 \vee \ldots \vee L_k$, set $\vartheta(n', n'') := \vartheta(n', n'') \cup \{\sim L_1, \ldots, \sim L_k\}$ where $\sim L_i$ is $\neg p$ if $L_i \equiv p \in \mathcal{V}$ is an unnegated proposition letter, $\sim L_i$ is p if $L_i \equiv \neg p$ is a negated proposition letter, $\sim\mathsf{t} = \mathsf{f}$, and $\sim\mathsf{f} = \mathsf{t}$. For every $(n', n'') \in N$ such that $((n', n''), (n^1, n^2)) \in S$, set $\vartheta(n', n'') := \vartheta(n', n'') \cup \{\mathsf{f}\}$.
- Mark the node (n^1, n^2) such that it will not be considered in future iteration steps.

Clearly, the algorithm computing the node labels of the conflict graph terminates. Optionally, the conflict graph T may now be "pruned" by performing the following step which eliminates nodes carrying an unsatisfiable label as well as trivial subgraphs:

States marked by the above procedure and nodes (except the initial node (n_0^1, n_0^2) of T) all of whose $(I \cup S)^*$-successors (including themselves) have an empty node label may be deleted from T. Nodes which have become unreachable by these deletions may as well be deleted. Notice that this step is optional: obviously, it does not change the set of temporal structures accepted by T, but reduces the size of T.

Sometimes, the *enriched node label* $\vartheta^+(n^1, n^2)$ of a node (n^1, n^2) in the conflict graph T of two transition graphs T^1 and T^2 will be referred to which is simply defined as

$$\vartheta^+(n^1, n^2) := \vartheta(n^1, n^2) \cup \vartheta^1(n^1) \cup \vartheta^2(n^2).$$

♠

The reader may wonder why the algorithm presented here is not symmetrical in treating the given transition graphs T^1 and T^2. In fact, fully symmetrical constructions may be given which produce equivalent conflict graphs w.r.t. the set of accepted temporal structures. It is merely for technical convenience in the proofs to follow that an unsymmetrical construction has been adopted here.

Example 4.26 As an example of the construction of a conflict graph, consider the (extended) program clauses

$$\begin{array}{rcl} \varphi & \equiv & r_1 \rightarrow (p \wedge q) \;\mathsf{unless}\; r_2 \quad \equiv \quad r_1 \rightarrow \nu X.\neg r_2 \rightarrow (p \wedge q \wedge \circ X) \\ \text{and} \quad \psi & \equiv & (\neg q \;\mathsf{atnext}\; p) \;\mathsf{atnext}\; r_3 \quad \equiv \quad \nu Y.(\neg r_3 \rightarrow \circ Y) \;\wedge\; (r_3 \rightarrow \\ & & \qquad\qquad \nu X.(p \rightarrow \neg q) \wedge (\neg p \rightarrow \circ X)). \end{array}$$

Regular and normal transition graphs S and T representing φ and ψ as well as the conflict graph of S and T are shown in figure 4.8. (Nontrivial combined

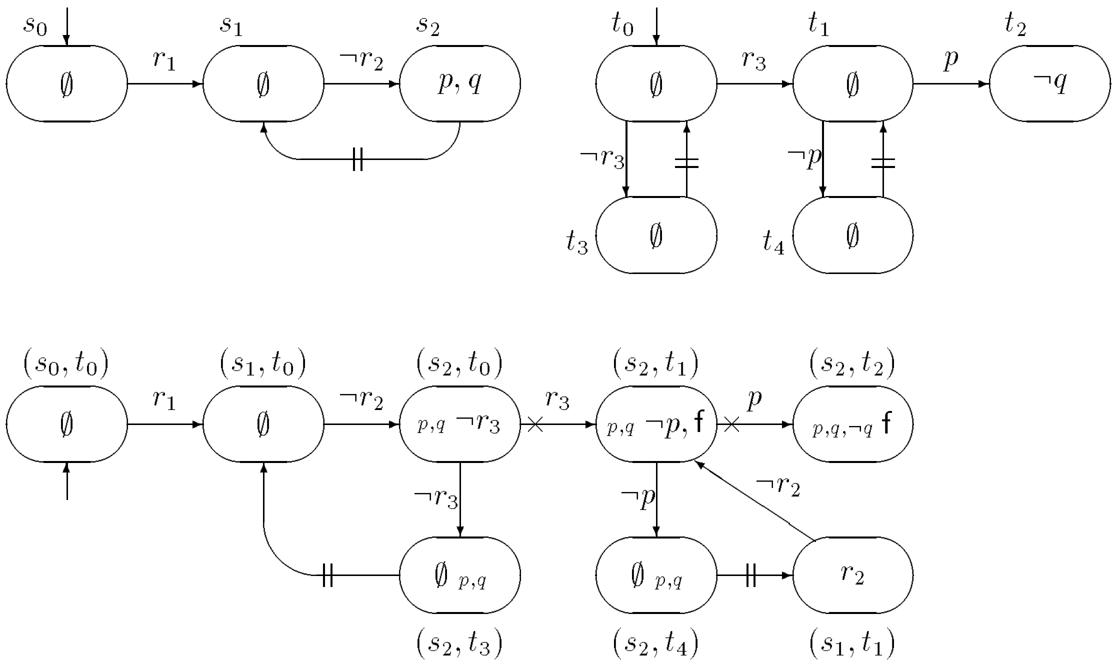


Figure 4.8: Example for the construction of a conflict graph

node labels of the original transition graphs are shown in the corresponding nodes of the conflict graph in small typeface.) Notice how the inconsistency detected at node (s_2, t_2) propagates back causing the label of (s_2, t_1) to become inconsistent. The optional pruning step would cut off the entire subgraph starting at node (s_2, t_1). The extended program clause corresponding to the (pruned) conflict graph can be constructed by algorithm 4.13, yielding (after simplification) the clause

$$r_1 \rightarrow (\nu X.\neg r_2 \rightarrow \neg r_3 \wedge \circ X) \equiv r_1 \rightarrow \neg r_3 \;\mathsf{unless}\; r_2.$$

It is the purpose of conflict graphs to identify inconsistent state combinations (represented by the combined labels of all nodes contained in the markings of the different transition graphs representing a νTL program) and to state the relevant restrictions ensuring that such inconsistencies never arise during the execution of the given program. What is required to guarantee a successful execution of the program is that every valuation of the environment components can be complemented by assignments to the module components in such a way that the resulting temporal structure is a model of every program clause.

The following lemmas will establish that the conditions expressed by conflict graphs are necessary and sufficient for this purpose. The first lemma states that the marking of the conflict graph over any structure is contained in the "product" of the markings of the original transition graphs.

Lemma 4.27 *Let T^1 and T^2 be two regular and normal transition graphs, and let T be the conflict graph of T^1 and T^2. Let $\mathcal{K}$ be a temporal structure and denote by M, M^1 and M^2, respectively, the markings of T, T^1 and T^2, resp., over $\mathcal{K}$. Then for every $i \in I\!N$ and all nodes $n^1 \in N^1$ and $n^2 \in N^2$, it holds that*

$$(n^1, n^2) \in m_i \quad \Longrightarrow \quad n^1 \in m_i^1 \text{ and } n^2 \in m_i^2.$$

Proof: by induction on the definition of M:

- For the initial node (n_0^1, n_0^2) of T, it holds by definition of M^1 and M^2 that $n_0^1 \in m_0^1$ and $n_0^2 \in m_0^2$.
- Assume that $(n^1, n^2) \in m_i$ because of some $(n', n'') \in m_i$ where $((n', n''), (n^1, n^2)) \in I$, and $\mathcal{K}_i \models \zeta((n', n''), (n^1, n^2))$. The induction hypothesis implies that $n' \in m_i^1$ and $n'' \in m_i^2$, and by the construction of T, either $n' = n^1$, $(n'', n^2) \in I^2$ and $\zeta((n', n''), (n^1, n^2)) = \zeta^2(n'', n^2)$, or $n'' = n^2$, $(n', n^1) \in I^1$ and $\zeta((n', n''), (n^1, n^2)) = \zeta^1(n', n^1)$. In either case, it follows that $n^1 \in m_i^1$ and $n^2 \in m_i^2$.
- Assume $(n', n'') \in m_i$ and $((n', n''), (n^1, n^2)) \in S$. Then by the induction hypothesis, $n' \in m_i^1$ and $n'' \in m_i^2$, and by the construction of T, $(n', n^1) \in S^1$ and $(n'', n^2) \in S^2$, hence by the definition of M^1 and M^2 it follows that $n^1 \in m_{i+1}^1$ and $n^2 \in m_{i+1}^2$.

♠

As a consequence of this lemma, a temporal structure will be accepted by the conflict graph if it is accepted by both the original graphs:

Lemma 4.28 *Let T, T^1 and T^2 be defined as before. Every structure which is accepted by both T^1 and T^2 is also accepted by T.*

Proof: Let $\mathcal{K}$ be a temporal structure which is accepted by T^1 and T^2, and denote by M, M^1 and M^2 the markings of T, T^1 and T^2, respectively, over $\mathcal{K}$. We will prove by induction on the number of iterations performed by the algorithm computing the node labelling of T that for every $i \geq 0$, $\mathcal{K}_i \models \vartheta(n^1, n^2)$ for every node $(n^1, n^2) \in m_i$. The assertion is trivial at the beginning of the algorithm computing the node labels $\vartheta(n^1, n^2)$ of the conflict graph. For the induction step, assume that it holds for all literals inserted into $\vartheta(n', n'')$ where $(n', n'') \in m_i$ for some $i \geq 0$ in earlier iterations, and that $(n^1, n^2) \in N$ has been chosen at the beginning of the current iteration step. Then $\vartheta^1(n^1) \cup \vartheta^2(n^2) \cup \vartheta(n^1, n^2)$ is inconsistent (where $\vartheta(n^1, n^2)$ denotes the current value of the node label of $(n^1, n^2) \in N$) which implies that $(n^1, n^2) \notin m_i$ for every $i \geq 0$:

Otherwise, by lemma 4.27 it would follow that $n^1 \in m_i^1$ and $n^2 \in m_i^2$, hence $\mathcal{K}_i \models \vartheta^1(n^1)$ and $\mathcal{K}_i \models \vartheta^2(n^2)$ because both original transition graphs accept $\mathcal{K}$. But then, $\mathcal{K}_i \not\models \vartheta(n^1, n^2)$ contradicting the induction hypothesis. Hence, setting $\vartheta(n^1, n^2) := \vartheta(n^1, n^2) \cup \{\mathsf{f}\}$ preserves the assertion. Now, $(n^1, n^2) \notin m_i$ for every $i \geq 0$ implies that $(n', n'') \notin m_i$ holds for every node (n', n'') such that $((n', n''), (n^1, n^2)) \in S$ and every $i \geq 0$, i.e. setting $\vartheta(n', n'') := \vartheta(n', n'') \cup \{\mathsf{f}\}$ for every such node (n', n'') similarly preserves the assertion. Finally, for every node $(n', n'') \in m_i$ for some $i \geq 0$ such that $((n', n''), (n^1, n^2)) \in I$ (assuming $\zeta((n', n''), (n^1, n^2)) = L_1 \vee \ldots \vee L_k$), it follows from $(n^1, n^2) \notin m_i$ that $\mathcal{K}_i \not\models \zeta((n', n''), (n^1, n^2))$, i.e. $\mathcal{K}_i \models \sim L_1, \ldots, \mathcal{K}_i \models \sim L_k$, establishing the assertion for the remaining assignments occurring in the iteration step considered. ♠

By lemma 4.28, acceptance by the conflict graph is a necessary condition for any temporal structure to be accepted by the original transition graphs. The following lemmas will establish that the acceptance of the conflict graph of two regular, normal, complete and monotonic transition graphs by some temporal structure $\mathcal{K}$ ensures that no two nodes of the original transition graphs carrying labels whose union would be inconsistent can be contained in some sets m_i^1, m_i^2 where M^1 and M^2 are the markings of the original transition graphs over $\mathcal{K}$.

Lemma 4.29 *Let T^1 and T^2 be regular, normal and complete transition graphs, and T be the conflict graph of T^1 and T^2. Let $\mathcal{K}$ be a temporal structure and σ be the state sequence of $\mathcal{K}$, and denote by M^1, M^2 and M the markings of T^1, T^2 and T over $\mathcal{K}$, respectively. Let, for some $i \geq 0$, $n' \in m_i^1$, $n'' \in m_i^2$ and $\pi' = (n'_0, \ldots n'_k)$ be a fullpath for n' and $\sigma[i]$, and $\pi'' = (n''_0, \ldots, n''_l)$ be a fullpath for n'' and $\sigma[i]$. Then there are $0 \leq u \leq k$ and $0 \leq v \leq l$ such that $(n'_u, n''_v) \in m_i$.*

Proof: by simultaneous induction on the definition of M^1 and M^2:

- If $i = 0$, and $n' = n_0^1$ and $n'' = n_0^2$ are the initial nodes of T^1 and T^2, respectively, then $(n', n'') = (n'_0, n''_0) \in m_i$ holds by the definition of M.
- Assume that the assertion holds for nodes $n_1 \in m_i^1$ and $n_2 \in m_i^2$ (for some $i \geq 0$). It has to be shown for the nodes $n' \in I^1(n_1)$ where $\mathcal{K}_i \models \zeta^1(n_1, n')$ and n_2 as well as for n_1 and the nodes $n'' \in I^2(n_2)$ where $\mathcal{K}_i \models \zeta^2(n_2, n'')$.

  Let $(n_1, n') \in I^1$ be an internal edge of T^1 such that $\mathcal{K}_i \models \zeta^1(n_1, n')$, and let $\pi' = (n'_0, \ldots, n'_k)$ and $\pi'' = (n''_0, \ldots, n''_l)$ be fullpaths for n' and $\sigma[i]$ and n_2 and $\sigma[i]$, respectively. Consider the fullpath $\pi^1 = n_1 \circ \pi'$ for n_1 and $\sigma[i]$ which passes through n'. By the induction hypothesis, there is some n''_v contained in π'' such that either $(n'_u, n''_v) \in m_i$ for some node n'_u appearing in the path π', in which case the assertion holds for n' as well, or $(n_1, n''_v) \in m_i$. But then, since $I^1(n_1) \neq \emptyset$ (because of $(n_1, n') \in I^1$), it follows from the construction of T that $((n_1, n''_v), (n', n''_v)) \in I$ and $\zeta((n_1, n''_v), (n', n''_v)) = \zeta^1(n_1, n')$. By $\mathcal{K}_i \models \zeta(n_1, n')$, $(n', n''_v) \in m_i$ which again establishes the assertion for n'.

Let now $(n_2, n'') \in I^2$ where $\mathcal{K}_i \models \zeta^2(n_2, n'')$, and let $\pi' = (n'_0, \ldots, n'_k)$ and $\pi'' = (n''_0, \ldots, n''_l)$ be fullpaths for n_1 and $\sigma[i]$ and n'' and $\sigma[i]$, respectively. By a repeated application of the previous argument, it follows that either $(n'_k, n''_v) \in m_i$ for some $0 \leq v \leq l$ in which case we are done, or that $(n'_k, n_2) \in m_i$. Since T^1 is complete and π' is a fullpath for n_1 and $\sigma[i]$ it follows in the latter case that $I^1(n'_k) = \emptyset$, hence by the construction of T, $((n'_k, n_2), (n'_k, n'')) \in I$ and $\zeta((n'_k, n_2), (n'_k, n'')) = \zeta^2(n_2, n'')$ which implies, similarly as above, that $(n'_k, n'') \in m_i$.

- Assume the assertion holds for $n_1 \in m^1_i$ and $n_2 \in m^2_i$, and let $(n_1, n') \in S^1$, $(n_2, n'') \in S^2$. Since T^1 and T^2 are regular, it follows that $I^1(n_1) = \emptyset$ and $I^2(n_2) = \emptyset$, hence both fullpaths π^1 for n_1 and $\sigma[i]$ and π^2 for n_2 and $\sigma[i]$ are trivial. Hence, the induction hypothesis yields that $(n_1, n_2) \in m_i$, and by the construction of T, it follows that $((n_1, n_2), (n', n'')) \in S$, hence $(n', n'') \in m_{i+1}$ which proves the assertion. ♠

As a consequence of this lemma, if T^1 and T^2 are also monotonic, no pair of nodes contained in corresponding states of the markings of the original transition graphs over some temporal structure can carry inconsistent labels if the conflict graph accepts the structure:

Lemma 4.30 *Let T^1 and T^2 be two normal, regular, complete and monotonic transition graphs and T be the conflict graph of T^1 and T^2. Let $\mathcal{K}$ be some temporal structure which is accepted by T, and denote by $M = (m_0, m_1, \ldots)$, $M^1 = (m^1_0, m^1_1, \ldots)$ and $M^2 = (m^2_0, m^2_1, \ldots)$ the markings of T, T^1 and T^2 over $\mathcal{K}$, respectively. Then for no $i \geq 0$ there are nodes $n^1 \in m^1_i$, $n^2 \in m^2_i$ such that the union $\vartheta^1(n^1) \cup \vartheta^2(n^2)$ of their labels contains a complementary pair $p, \neg p$ of literals.*

Proof: Again, the state sequence of the temporal structure $\mathcal{K}$ is denoted by σ. Let $n^1 \in m^1_i$ and $n^2 \in m^2_i$, for some $i \geq 0$, and let $\pi^1 = (n^1_0, \ldots, n^1_k)$ and $\pi^2 = (n^2_0, \ldots, n^2_l)$ be fullpaths for n^1 and $\sigma[i]$ and n^2 and $\sigma[i]$, respectively. By lemma 4.29, there are $0 \leq u \leq k$ and $0 \leq v \leq l$ such that $(n^1_u, n^2_v) \in m_i$. Since $\mathcal{K}$ is accepted by T, $\mathsf{f} \notin \vartheta(n^1_u, n^2_v)$. But then, by the construction of T, it follows that the enriched node label $\vartheta^+(n^1_u, n^2_v) = \vartheta^1(n^1_u) \cup \vartheta^2(n^2_v) \cup \vartheta(n^1_u, n^2_v)$ does not contain a complementary pair of literals. Since T^1 and T^2 were assumed to be monotonic transition graphs, $\vartheta^1(n^1) \subseteq \vartheta^1(n^1_u)$ and $\vartheta^2(n^2) \subseteq \vartheta^2(n^2_v)$, hence the assertion follows. ♠

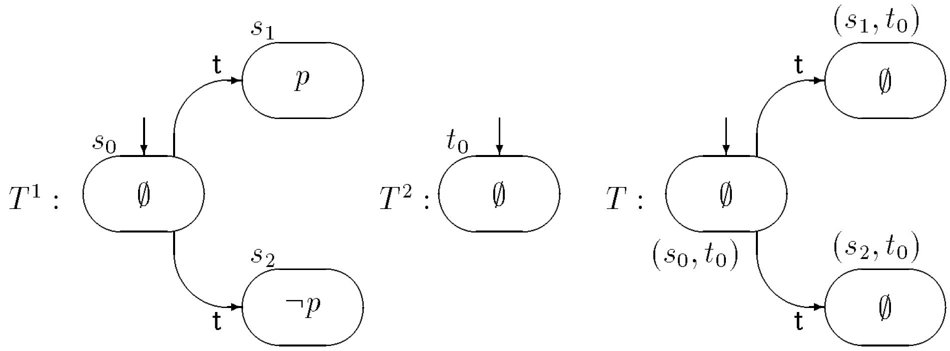


Figure 4.9: Inconsistency of T^1 is not recognized by T

Notice, however, that the acceptance of $\mathcal{K}$ by the conflict graph T is not enough to ensure acceptance by both T^1 and T^2. As a trivial example, consider the situation represented in figure 4.9: T^1 does not accept any temporal structure, whereas T^2 as well as the conflict graph T of T^1 and T^2 accept every temporal structure.

The conflict graph of T^1 and T^2 can only discover possible inconsistencies *between* T^1 and T^2, not those hidden *within* a single transition graph. However, such inconsistencies can be discovered by a conflict graph of two copies of the same transition graph. Let us introduce some further notation which also accounts for the environment assumptions referred to in section 4.1:

An *environment structure* $\mathcal{E}$ is an ω-sequence of valuations $(\varepsilon_0, \varepsilon_1, \ldots)$ where each $\varepsilon_i : \mathcal{V}_E \to 2^{\{()\}}$ assigns some truth value to every environment component $p \in \mathcal{V}_E$. A temporal structure $\mathcal{K} = (\sigma_0, \sigma_1, \ldots)$ is called an *extension* of an environment structure $\mathcal{E}$ (denoted by $\mathcal{K} \sqsupseteq \mathcal{E}$) if $\sigma_i(p) = \varepsilon_i(p)$ for every $i \geq 0$ and every environment component $p \in \mathcal{V}_E$. A set $\mathcal{U}$ of environment structures can be taken to represent all admissible behaviors of the environment according to the environment assumptions adopted by the system developer.

Definition 4.31 (compatibility) *Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be a propositional νTL program (i.e. a set of program clauses) and $\mathcal{U}$ a set of environment structures. Denote by $T_1, \ldots, T_n$ regular, normal, complete and monotonic versions of the transition graphs associated with $\varphi_1, \ldots, \varphi_n$, respectively. Φ is called $\mathcal{U}$-*compatible *if, for every $0 \leq k < n$ and every $\mathcal{E} \in \mathcal{U}$, every model $\mathcal{K} \sqsupseteq \mathcal{E}$ of $\varphi_1, \ldots, \varphi_k$ is accepted by the conflict graphs of T^{k+1} and T^j for every $j = 1, \ldots, k+1$.*

In practice, $\mathcal{U}$ will be given implicitly by some νTL formula Ψ stating the environment assumptions. The relevant condition for $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ to be compatible w.r.t. Ψ can in this case be stated in the following form: For every $0 \leq k < n$, the νTL formulas associated with the conflict graphs of T^{k+1} and T^j (for every $j = 1, \ldots, k+1$) by algorithm 4.13 follow from Ψ and $\{\varphi_1, \ldots, \varphi_k\}$. These two formulations of the compatibility criterion are equivalent due to the lemmas 4.11 and 4.15.

4.5 Interpreting propositional νTL programs

$\mathcal{U}$-compatible νTL programs Φ admit the construction of a model $\mathcal{K} \sqsupseteq \mathcal{E}$ for every "admissible" environment structure $\mathcal{E} \in \mathcal{U}$. Let us first give the construction algorithm and subsequently prove its correctness (in the sequel, $\mathcal{V}$ denotes the finite set of proposition letters appearing in Φ):

Algorithm 4.32 (interpreter for propositional νTL programs)
Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a propositional νTL program and $\mathcal{E} = (\varepsilon_0, \varepsilon_1, \ldots)$ be an environment structure. Denote by $T^k = (N^k, I^k, S^k, \vartheta^k, \zeta^k, n_0^k)$ (for $1 \leq k \leq q$) the normal transition graph associated with the program clause φ^k by algorithm 4.9 and lemma 4.17. Construct the temporal structure $\mathcal{K}(\Phi, \mathcal{E}) = (\sigma_0, \sigma_1, \ldots)$ as follows (during the construction of σ_i, the pseudo-value $\sigma_i(p) = \perp$ denotes that $\sigma_i(p)$ is still undefined):

A *configuration* of $T^1, \ldots, T^q$ consists of a q-tuple $\kappa = (\nu^1, \ldots, \nu^q)$ of sets $\nu^k \subseteq N^k$ of nodes of the transition graphs and a set Exp representing the edges of the transition graphs which still have to be inspected. By $L \in \zeta^k(n, n')$ we will denote that the literal L appears as a disjunct in the edge label $\zeta^k(n, n')$ (recall that all T^k are normal transition graphs). The algorithm defines a sequence of configurations $(\kappa_i^0, \kappa_i^1, \ldots)$ for every state σ_i of the temporal structure $\mathcal{K}(\Phi, \mathcal{E})$ under construction:

Initialization: Set $i := 0$ and $j := 0$ and define the initial configuration $\kappa_0^0 := (\{n_0^1\}, \ldots, \{n_0^q\})$.

State construction: Assume that $\kappa_i^0 = (\nu^1, \ldots, \nu^q)$ is the initial configuration for the state σ_i and let

$$Exp := \bigcup_{k=1}^{q} \{(n, n', L) \mid n \in \nu^k, (n, n') \in I^k, L \in \zeta^k(n, n')\}$$

be the initial set of edges to explore. Define

$$\sigma_i(p) := \left\{ \begin{array}{cl} \varepsilon_i(p) & \text{if } p \in \mathcal{V}_E \text{ is an environment component} \\ \{()\} & \text{if } p \in \vartheta^k(n) \text{ for some } n \in \nu^k (1 \leq k \leq q) \\ \emptyset & \text{if } \neg p \in \vartheta^k(n) \text{ for some } n \in \nu^k (1 \leq k \leq q) \\ \perp & \text{else} \end{array} \right\}.$$

Complete the construction of σ_i as follows:

1. While $Exp \neq \emptyset$, iterate the following steps:
   (a) Set $j := j+1$. Choose some $(n_k, n'_k, L) \in Exp$ (where $n_k, n'_k \in N^k$) such that L is minimal among all literals appearing in Exp, i.e. for no $(n_l, n'_l, L') \in Exp$ it holds that $L' < L$ w.r.t. the dependency order $<$ on literals. If $L \equiv (\neg)p$ for some $p \in \mathcal{V}$ and $\sigma_i(p) = \perp$ (i.e. $\sigma_i(p)$ is yet "undefined"), then set $\sigma_i(p) := \{()\}$ or $\sigma_i(p) := \emptyset$.
   (b) If, for some $p \in \mathcal{V}$, $L \equiv p$ and $\sigma_i(p) = \{()\}$, or if $L \equiv \neg p$ and $\sigma_i(p) = \emptyset$, or if $L \equiv \mathtt{t}$, then $\kappa_i^j := (\nu^1, \ldots, \bar{\nu}^k, \ldots, \nu^q)$ where $\bar{\nu}^k = \nu^k \cup \{n'_k\}$. In this case, define $\sigma_i(p) := \{()\}$ for every $p \in \vartheta^k(n'_k)$ and $\sigma_i(p) := \emptyset$ for every p such that $\neg p \in \vartheta^k(n'_k)$. Delete all elements of the form (n''_k, n'_k, L') representing internal edges of T^k leading to n'_k from Exp and set
   $$Exp := Exp \cup \{(n'_k, n''_k, L') \mid (n'_k, n''_k) \in I^k, L' \in \zeta^k(n'_k, n''_k)\}.$$
   Else, $\kappa_i^j := \kappa_i^{j-1}$ and $Exp := Exp \setminus \{(n_k, n'_k, L)\}$.
2. For all $p \in \mathcal{V}$ such that $\sigma_i(p) = \perp$, define $\sigma_i(p) := \{()\}$ or $\sigma_i(p) := \emptyset$.
3. Assume that the final configuration for the construction of σ_i (i.e. the first configuration for σ_i where $Exp = \emptyset$) is $\kappa_i^j = (\nu^1, \ldots, \nu^q)$. Set $\kappa_{i+1}^0 := (\bar{\nu}^1, \ldots, \bar{\nu}^q)$ where $\bar{\nu}^k = \bigcup_{n \in \nu^k} S(n)$. Set $i := i+1$ and $j := 0$ and repeat the state construction starting with the new configuration.

Notice that the algorithm can work incrementally: Every state σ_i can be produced as soon as ε_i has been input; moreover, backtracking never occurs during the model construction. If Φ is a $\mathcal{U}$-compatible νTL program, this algorithm actually computes (prefixes of) a model of Φ for every environment structure $\mathcal{E} \in \mathcal{U}$ satisfying the environment assumptions $\mathcal{U}$:

Theorem 4.33 *Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a $\mathcal{U}$-compatible propositional νTL program where $\mathcal{U}$ is some set of environment structures, and let $\mathcal{E} \in \mathcal{U}$. Then $\mathcal{K}(\Phi, \mathcal{E})$ as defined in algorithm 4.32 is well-defined and is a model of Φ.*

Proof: Let us first prove that $\mathcal{K}(\Phi, \mathcal{E})$ is a model of Φ if it is well-defined. Let $M^k = (m_0^k, m_1^k, \ldots)$ denote the marking of T^k over $\mathcal{K}(\Phi, \mathcal{E})$. A straightforward induction on i and j shows that for every $i \geq 0$, every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$ and every $k = 1, \ldots, q$, $\nu^k \subseteq m_i^k$ and $m_i^k \subseteq \nu^k \cup \bigcup\{(I^k)^+(n) \mid (n, n', L) \in Exp \text{ where } n \in N^k\}$. In particular, for those configurations where $Exp = \emptyset$, it follows that $\nu^k = m_i^k$ and, by construction of $\mathcal{K}(\Phi, \mathcal{E})$ and the assumption that it is well-defined, for every $n \in \nu^k$ it follows that $\sigma_i \models L$ for every literal $L \in \vartheta(n)$. (Notice that the node labels $\vartheta(n)$, for every $n \in N^k$, can only contain literals built from proposition letters $p \in \mathcal{V}_\Pi$ since φ^k is a proper program clause. In particular, $\mathsf{f} \notin \vartheta(n)$.) Thus, $\mathcal{K}(\Phi, \mathcal{E})$ is accepted by every transition graph T^k, hence $\mathcal{K}(\Phi, \mathcal{E}) \models \varphi^k$ for every $1 \leq k \leq q$ (using lemma 4.11).

To show that under the assumptions stated above, $\mathcal{K}(\Phi, \mathcal{E})$ is well-defined, observe first that by step 2, every proposition letter is eventually assigned some proper value $\sigma_i(p)$ for every $i \geq 0$. Moreover, no conflicts can occur in the assignments of truth values to environment components since literals formed from such proposition letters do not appear in the node labels of any transition graph T^k. Besides the termination of the construction of every single state σ_i, three points remain to be shown (the notation is as in algorithm 4.32):

(a) For every $i \geq 0$, every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q)$ and all $1 \leq k, l \leq q$: If $p \in \vartheta^k(n)$ for some $n \in \nu^k$ then for no configuration $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^n)$ where $m \geq j$ there is some node $n' \in \bar{\nu}^l$ such that $\neg p \in \vartheta^l(n')$.

(b) For every $i \geq 0$, every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^n)$ and all $1 \leq k, l \leq q$: If $\neg p \in \vartheta^k(n)$ for some $n \in \nu^k$ then for no configuration $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^n)$ where $m \geq j$ there is some node $n' \in \bar{\nu}^l$ such that $p \in \vartheta^l(n')$.

(c) For every $i \geq 0$: If $\sigma_i(p)$ has been defined in step (1a) then for no configuration $\kappa_i^j = (\nu^1, \ldots, \nu^n)$ and no $1 \leq k \leq q$ there is some node $n \in \nu^k$ such that $p \in \vartheta(n)$ or $\neg p \in \vartheta(n)$.

Proof sketches for (a), (b) and (c) are as follows:

(c) The triple (n_k, n'_k, L) leading to the definition of $\sigma_i(p)$ by step (1a) has been chosen such that $L \equiv (\neg)p$ is minimal w.r.t. all candidate edges represented by the set Exp. No set ν^l of the current or an earlier configuration κ_i^j contains a node in whose label appears either p or $\neg p$ — else, $\sigma_i(p)$ would have received a proper value in an earlier iteration. Clearly, every node $\bar{n}_l \in \nu^l$ contained in some future configuration κ_i^m is an $(I^l)^+$-successor of some node n_l such that Exp contains an element of the form (n_l, n'_l, L'). By the choice of (n_k, n'_k, L), $L' \not< L$ for every such triple. But by lemma 4.12, this would have to hold if $\vartheta(\bar{n}_l)$ contained p or $\neg p$.

(a,b) Assume, to the contrary, that for some (smallest) pair $1 \leq k, l \leq q$ there is some (smallest) $i \geq 0$ such that there exist configurations $\kappa_i^j = (\nu^1, \ldots, \nu^q)$ and $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^q)$ (where $m \geq j$) and nodes

$n \in \nu^k, n' \in \bar{\nu}^l$ such that $p \in \vartheta^k(n)$ and $\neg p \in \vartheta^l(n')$ (that $\neg p \in \vartheta^k(n)$ and $p \in \vartheta^l(n')$, resp.). Only case (a) will be considered here, since (b) is symmetrical. W.l.o.g., assume $k \leq l$. Now consider the program $\Phi' = \{\varphi^1, \ldots, \varphi^{l-1}\}$. Apply algorithm 4.32 to construct the structure $\mathcal{K}(\Phi', \mathcal{E}) = (\sigma'_0, \sigma'_1, \ldots)$ where for every $s < i$ and every proposition letter $q \in \mathcal{V}$, the valuation $\sigma'_s(q)$ is chosen as $\sigma'_s(q) = \sigma_s(q)$ whenever it would be "undefined" in $\mathcal{K}(\Phi', \mathcal{E})$ (i.e. in the assignments of steps (1a) and 2) and, similarly, $\sigma'_i(q) = \sigma_i(q)$ for every $q < p$ if $\sigma'_i(q)$ is undefined. By the choice of k and l, the structure $\mathcal{K}(\Phi', \mathcal{E})$ is well-defined, hence (see above) it is a model of Φ' and obviously it is an extension of $\mathcal{E}$. Moreover, $\sigma'_s(q) = \sigma_s(q)$ for every $s < i$ and $q \in \mathcal{V}$ as well as $\sigma'_i(q) = \sigma_i(q)$ for $q < p$. Since Φ is $\mathcal{U}$-compatible, it follows that $\mathcal{K}(\Phi', \mathcal{E})$ is accepted by the conflict graph of regular, complete and monotonic versions of T^k and T^l. Lemma 4.30 implies (using lemmas 4.16, 4.18 and 4.19) that not both $n \in m^k_i$ and $n' \in m^l_i$ can hold (where M^k and M^l denote the markings of T^k and T^l over $\mathcal{K}(\Phi', \mathcal{E})$, resp.). On the other hand, it has been remarked above that the configurations κ^j_i approximate the markings of the transition graphs $T^1, \ldots, T^q$ over the structure under construction, and since $\mathcal{K}(\Phi, \mathcal{E})$ and $\mathcal{K}(\Phi', \mathcal{E})$ agree in the relevant prefix, $n \in m^k_i$ and $n' \in m^l_i$ would follow from $n \in \bar{\nu}^k$ and $n' \in \bar{\nu}^l$, establishing a contradiction.

To prove the termination of the construction of every single state, it suffices to observe that the size of the "unexplored parts" of the transition graphs measured as

$$|\bigcup_{k=1}^{q} \{(I^k)^+(n) \mid n \in N^k, \exists n', L : (n, n', L) \in Exp\}| + |Exp|$$

decreases with every iteration step.

In particular, the construction of any single state of the model $\mathcal{K}(\Phi, \mathcal{E})$ is linear in the combined size of the normal transition graphs associated with the program clauses in Φ, measured as the number of edges plus the size of the node labels $\vartheta^k(n)$ (which in turn is linear in the sum of the lengths of the program clauses measured as the number of connectives if every antecedent is in conjunctive normal form). ♠

Example 4.34 As an example for a propositional νTL program, consider a simple resource allocator which grants its resource in response to requests from two client processes Π_1 and Π_2, ensuring exclusive access to the resource. The client processes signal their requests via the environment components (as seen from the allocator) R_1 and R_2, respectively, each client process resetting its request only after it has been allocated the resource and is willing to release it. The allocator uses the module components G_1 and G_2 to signal that the resource is granted to Π_1 or Π_2, respectively. Define the dependency order on $\mathcal{V}_\Pi = \{G_1, G_2\}$ by $G_2 < G_1$. A propositional νTL program implementing the allocator module is given by the following four program clauses:

$$
\begin{array}{lcll}
\varphi^1 & \equiv & \Box(R_1 \rightarrow & ((G_1 \;\mathsf{unless}\; \neg R_1) \;\mathsf{atnext}\; \neg G_2) \\
 & & & \wedge ((\neg G_1 \;\mathsf{unless}\; (R_1 \wedge \neg G_2)) \;\mathsf{atnext}\; \neg R_1)) \\
\varphi^2 & \equiv & \Box(R_2 \rightarrow & (\bigcirc(G_2 \;\mathsf{unless}\; \neg R_2) \;\mathsf{atnext}\; \neg G_1) \\
 & & & \wedge \;\; ((\neg G_2 \;\mathsf{upto}\; (R_2 \wedge \neg G_1)) \;\mathsf{atnext}\; \neg R_2)) \\
\varphi^3 & \equiv & \neg G_1 \;\mathsf{unless}\; (R_1 \wedge \neg G_2) & \\
\varphi^4 & \equiv & \neg G_2 \;\mathsf{upto}\; (R_2 \wedge \neg G_1) &
\end{array}
$$

where the temporal operator upto is defined by

$$F \;\mathsf{upto}\; G \equiv \nu X.F \wedge (\neg G \rightarrow \bigcirc X) \equiv (F \;\mathsf{unless}\; G) \wedge (F \;\mathsf{atnext}\; G).$$

Replacing the derived operators by their formal νTL definitions, it may be verified that the formulas $\varphi^1, \ldots, \varphi^4$ are proper program clauses according to definition 4.2. For example, φ^1 becomes

$$
\begin{array}{ll}
\varphi^1 \equiv \nu X. \; (R_1 \rightarrow & (\nu Y. \; \neg G_2 \rightarrow (\nu Z.R_1 \rightarrow G_1 \wedge \bigcirc Z) \wedge \\
 & \qquad\qquad G_2 \rightarrow \bigcirc Y) \wedge \\
 & (\nu U. \; \neg R_1 \rightarrow (\nu V.\neg R_1 \vee G_2 \rightarrow \neg G_1 \wedge \bigcirc V) \wedge \\
 & \qquad\qquad R_1 \rightarrow \bigcirc U)) \\
\qquad \wedge \bigcirc X &
\end{array}
$$

The clauses φ^1 and φ^2 implement the allocator's strategy for reacting to service and release requests by the client processes Π_1 and Π_2, respectively. On the other hand, φ^3 and φ^4 initialize the values of the allocator's state components G_1 and G_2 until the first request by Π_1 or Π_2, respectively, can be served.

The asymmetry in the clauses favors Π_1 over Π_2 in the case of a simultaneous request by both clients. (Of course, this is a simple and unfair solution. For a better solution of a similar problem, witness the dining philosophers example in chapter 7.) Formally, observe that replacing φ^2 by a symmetric version of φ^1 would not yield a proper program clause according to definition 4.2. For instance, it would contain the subformula $\neg G_1 \rightarrow (\nu Z.R_2 \rightarrow G_2 \wedge \bigcirc Z)$ which would require $G_1 < G_2$ to hold for being an admissible pre-clause for a program clause.

Figure 4.10 displays normal transition graphs $T^1, \ldots, T^4$ associated with the program clauses $\varphi^1, \ldots, \varphi^4$. It is not hard to verify that the given program is $\mathcal{U}$-compatible w.r.t. any environment assumption $\mathcal{U}$. For example, all nodes

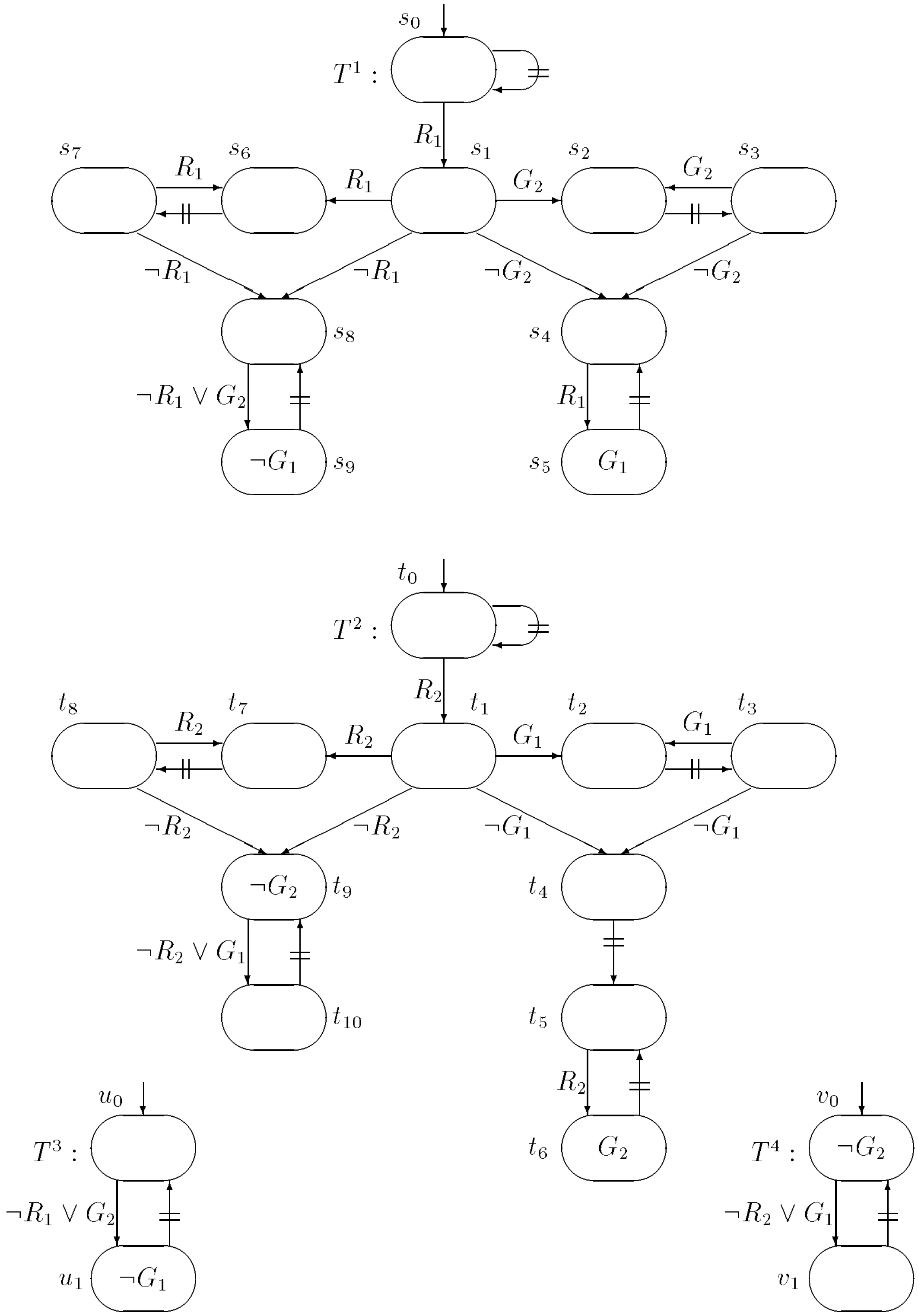


Figure 4.10: Transition graphs for the program clauses of example 4.34

of T^1 and T^3 containing G_1 in their node labels are reached only in states satisfying $R_1 \wedge \neg G_2$, whereas $\neg R_1 \vee G_2$ has to hold for reaching nodes which are labelled by $\neg G_1$.

Tables 4.1 and 4.2 show, respectively, the explicit construction of the first state and (in a more compact form) the construction of several later states of the structure $\mathcal{K}(\Phi, \mathcal{E})$ as defined by algorithm 4.32 for the prefix $(\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_5)$ of an environment structure $\mathcal{E}$ given in the following table (where '+' stands for $\{()\}$, i.e. "true" and '$-$' stands for $\emptyset$, i.e. "false").

i	0	1	2	3	4	5
$\varepsilon_i(R_1)$	+	+	$-$	$-$	$-$	+
$\varepsilon_i(R_2)$	$-$	+	+	+	$-$	+

An interpreter should represent the set Exp by an ordered list of triples w.r.t. the dependency order $<$. Such a representation has also been used in table 4.1. In table 4.2, only the first and last configurations for every state σ_i of $\mathcal{K}(\Phi, \mathcal{E})$ are displayed.

i	j	R_1	R_2	ν^1	ν^2	ν^3	ν^4	Exp	G_1	G_2
0	0	$+$	$-$	s_0	t_0	u_0	v_0	(s_0, s_1, R_1), (t_0, t_1, R_2), $(u_0, u_1, \neg R_1)$, $(v_0, v_1, \neg R_2)$, (u_0, u_1, G_2), (v_0, v_1, G_1)	$\perp$	$-$
0	1	$+$	$-$	s_0, s_1	t_0	u_0	v_0	(t_0, t_1, R_2), $(u_0, u_1, \neg R_1)$, $(v_0, v_1, \neg R_2)$, (s_1, s_6, R_1), $(s_1, s_8, \neg R_1)$, (u_0, u_1, G_2), (s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$, (v_0, v_1, G_1)	$\perp$	$-$
0	2	$+$	$-$	s_0, s_1	t_0	u_0	v_0	$(u_0, u_1, \neg R_1)$, $(v_0, v_1, \neg R_2)$, (s_1, s_6, R_1), $(s_1, s_8, \neg R_1)$, (u_0, u_1, G_2), (s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$, (v_0, v_1, G_1)	$\perp$	$-$
0	3	$+$	$-$	s_0, s_1	t_0	u_0	v_0	$(v_0, v_1, \neg R_2)$, (s_1, s_6, R_1), $(s_1, s_8, \neg R_1)$, (u_0, u_1, G_2), (s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$, (v_0, v_1, G_1)	$\perp$	$-$
0	4	$+$	$-$	s_0, s_1	t_0	u_0	v_0, v_1	(s_1, s_6, R_1), $(s_1, s_8, \neg R_1)$, (u_0, u_1, G_2), (s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$	$\perp$	$-$
0	5	$+$	$-$	s_0, s_1, s_6	t_0	u_0	v_0, v_1	$(s_1, s_8, \neg R_1)$, (u_0, u_1, G_2), (s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$	$\perp$	$-$
0	6	$+$	$-$	s_0, s_1, s_6	t_0	u_0	v_0, v_1	(u_0, u_1, G_2), (s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$	$\perp$	$-$
0	7	$+$	$-$	s_0, s_1, s_6	t_0	u_0	v_0, v_1	(s_1, s_2, G_2), $(s_1, s_4, \neg G_2)$	$\perp$	$-$
0	8	$+$	$-$	s_0, s_1, s_6	t_0	u_0	v_0, v_1	$(s_1, s_4, \neg G_2)$	$\perp$	$-$
0	9	$+$	$-$	s_0, s_1, s_6, s_4	t_0	u_0	v_0, v_1	(s_4, s_5, R_1)	$\perp$	$-$
0	10	$+$	$-$	s_0, s_1, s_6, s_4, s_5	t_0	u_0	v_0, v_1		$+$	$-$

Table 4.1: Construction of the first state of $\mathcal{K}(\Phi, \mathcal{E})$

i	R_1	R_2	ν^1	ν^2	ν^3	ν^4	G_1	G_2
1	+	+	s_0, s_7, s_4	t_0		v_0	$\perp$	$-$
1	+	+	$s_0, s_7, s_4, s_1, s_6, s_5$	t_0, t_1, t_7, t_2		v_0, v_1	+	$-$
2	$-$	+	s_0, s_7, s_4	t_0, t_8, t_3		v_0	$\perp$	$-$
2	$-$	+	s_0, s_7, s_4, s_8, s_9	$t_0, t_8, t_3, t_1, t_7, t_4$		v_0	$-$	$-$
3	$-$	+	s_0, s_6, s_8	t_0, t_8, t_5			$\perp$	$\perp$
3	$-$	+	s_0, s_6, s_8, s_9	$t_0, t_8, t_5, t_1, t_7, t_4, t_6$			$-$	+
4	$-$	$-$	s_0, s_7, s_8	t_0, t_8, t_5			$\perp$	$\perp$
4	$-$	$-$	s_0, s_7, s_8, s_9	$t_0, t_8, t_5, t_9, t_{10}$			$-$	$-$
5	+	+	s_0, s_8	t_0, t_9			$\perp$	$-$
5	+	+	$s_0, s_8, s_1, s_6, s_4, s_5$	$t_0, t_9, t_1, t_7, t_2, t_{10}$			+	$-$

Table 4.2: States $\sigma_1 - \sigma_5$ of $\mathcal{K}(\Phi, \mathcal{E})$ as generated by algorithm 4.32

Chapter 5

First-order νTL programs

5.1 First-order program clauses

Chapter 4 dealt with the development of (single modules of) finite-state reactive systems using propositional νTL. In particular, a definition of "executable" propositional νTL formulas was provided, together with a criterion of "compatibility" characterizing those sets of program clauses which allow the model construction algorithm to produce a model for every environment structure satisfying the environment assumptions. In general, the framework of propositional temporal logic may be found too weak or simply inconvenient for practical system development, being applicable only to systems with finite state spaces of bounded size. For example, whereas most synchronization algorithms employ a protocol using a (small) number of different states per process and can thus be specified using propositional temporal logics if the number of processes is fixed in advance (the simple resource allocator of example 4.34 being quite typical in this respect), the same algorithms cannot be described satisfactorily by propositional temporal logics for a finite but unbounded set of processes — even though the algorithms as well as their correctness proofs can usually be generalized. This chapter will therefore be concerned with extending the approach of executing certain νTL formulas to first-order logic. As a consequence of the incompleteness results for all but the most restricted first-order temporal logics given in chapter 3, program development becomes more complex in this framework. At least, no automatic solutions can be provided for the synthesis of modules implementing a given specification, whereas, in principle, this is possible for propositional specifications (cf. MANNA & WOLPER [39]), inefficient as the automatically generated implementation may be. Similarly, proofs required during the development process, such as proving that a refined specification implies the previous one or the proof of compatibility using the "proof-theoretic" formulation of this condition following definition 4.31, can in general no longer be found automatically, or even semi-automatically. However, this problem is not specific to the temporal logic approach to program development but is an issue for every sufficiently expressive formalism for the description of and the reasoning about potentially infinite state spaces (maybe it has not received the same degree of attention in other formalisms). On the other hand, if it is true that the idea of an algorithm does not depend on the actual size of the problem, it should be possible to generalize the corresponding correctness proof from a finite-state (propositional) version of the program to the first-order version — and sensibly chosen logical proof rules will support this generalization. In fact, the success of specification formalisms like UNITY [14] supports this view.

The state space of finite-state systems was represented in propositional νTL by assigning each state component (which could only assume values drawn from a fixed finite set) a finite list of boolean variables whose values in some state of a temporal structure represented the actual value of the state component in the corresponding state of the program execution. For (potentially) infinite-state programs, two technically different representations of state components are possible: First, *flexible variables* may be employed whose values in every state of the temporal structure agree with the values of the represented state components in the corresponding state of the execution of the program. Sets, lists, database relations etc. may be modelled more adequately by flexible *predicate symbols* whose extensions in every state of the temporal structure consist of the represented component's members in the corresponding execution state — recall the distinction between TLV and TLP logics in chapter 3. Both ways of modelling can coexist and will be allowed for first-order νTL programs. In both cases, the universe (data types of program variables, domains of database relations etc.) as well as functions operating on the objects are supplied by the underlying first-order structure which we will assume to be fixed outside of the program. During the execution of the program it must be ensured that all actual predicate extensions in every state are finite in order to be able to represent them on a (finite-state!) computer[1]. This requirement induces some subtle variable conditions for program clauses.

The interpreter will store the extension of a predicate in the current state, like a database relation, in the form of a set of object tuples. To ensure the effectiveness of an interpreter for first-order program clauses, it will be taken care

[1]It might be argued here that it is enough to be able to find a finitistic representation of the predicates' extensions, as, e.g., provided in PROLOG by non-ground literals. Such an approach, however, is only viable in the context of uninterpreted first-order structures, i.e. Herbrand universes (cf. the problems posed by arithmetical expressions appearing in PROLOG programs which can not be dealt with by the normal resolution and unification techniques).

that at every state there can only be a finite number of ground instances of program clauses which "fire", i.e. whose antecedents are true. Consequently (given infinite universes), it will not be possible to explicitly characterize the set of object tuples *not* contained in a predicate's extension at a given state by program clauses. Hence, a position similar to the closed-world assumption known from logic programming (cf. REITER [46] and LLOYD [35]) will be adopted for determining the current extensions of predicates: $(d_1, \ldots, d_n) \in \sigma_i(P)$ will hold only if there is some program clause explicitly forcing this relation to hold. But then there is no need for a pre-clause like $\neg P(t_1, \ldots, t_n)$, "writing" a negated prime formula (although such formulas could be given an operational interpretation as an explicit delete operation). Eliminating negative literals from pre-clauses essentially simplifies the compatibility condition for first-order program clauses because no inconsistencies caused by positive and negative occurrences of predicate symbols can be introduced. On the other hand, like in negation-free PROLOG programs, no explicit "negative" information stating that some object tuple is *not* contained in a relation can in this case be derived from the program text (viewed as a set of νTL formulas). Stated differently, declarative and operational semantics do not agree anymore which may well create problems during the correctness proof. Fortunately, first-order temporal logic is strong enough to characterize such a "minimal-model" interpreter strategy. This leads to the notion of *program completion*, transforming a given program to yield a set of νTL formulas whose logical semantics agrees with the operational semantics of the original program and which can be used in its place in the correctness proof. The necessary transformation can be carried out schematically for every program. Negative literals in antecedents, however, are allowed (if certain variable conditions are satisfied) and will be interpreted as tests.

The general observations which suggested the syntactical form of propositional program clauses remain valid in the first-order case. Let us now give formal definitions for antecedents and extended first-order νTL program clauses as subclasses of first-order νTL formulas (over some given language $\mathcal{L}$) similarly as these classes of formulas were defined in the propositional case. Simultaneously, sets $Bind(A)$ and $Open(P)$ of rigid subject variables which can be bound by the antecedent A or which are open in the pre-clause P are defined:

Definition 5.1 (antecedents)

- *Let t be a term. Define the set $Bind(t)$ of rigid subject variables which can possibly be bound by an occurrence of t in an antecedent by*

$$Bind(t) = \left\{ \begin{array}{ll} \{y\} & \textit{if } t \equiv y \in \mathcal{Y} \textit{ is a rigid subject variable} \\ \emptyset & \textit{else} \end{array} \right\}.$$

- *Atomic formulas $P(t_1, \ldots, t_m)$ and $t_1 = t_2$ as well as their negations and the propositional constants* t *and* f *are* literals. *The set of variables a literal can bind is defined by*

$$\begin{aligned}
Bind(P(t_1, \ldots, t_m)) &= (\bigcup_{i=1}^{m} Bind(t_i)) \setminus (\bigcup_{Bind(t_j)=\emptyset} FRV(t_j)), \\
Bind(t_1 = t_2) &= Bind(t_1) \setminus FRV(t_2), \\
Bind(\neg P(t_1, \ldots, t_m)) &= \emptyset, \quad Bind(t_1 \neq t_2) = \emptyset, \\
Bind(\mathsf{t}) &= \emptyset, \qquad\quad Bind(\mathsf{f}) = \emptyset.
\end{aligned}$$

- *Every literal is an* antecedent. *If A and B are antecedents, then $A \wedge B$ and $A \vee B$ are also antecedents where $Bind(A \wedge B) = Bind(A \vee B) = Bind(A) \cap Bind(B)$.*

Definition 5.2 (extended program clauses)

- *Every atomic formula of the form $P(t_1, \ldots, t_m)$ or $t_1 = t_2$ is a pre-clause where $Open(P(t_1, \ldots, t_m))$ and $Open(t_1 = t_2)$ are defined as the sets $FRV(P(t_1, \ldots, t_m))$ and $FRV(t_1 = t_2)$ of free rigid subject variables appearing in the formulas $P(t_1, \ldots, t_m)$ and $t_1 = t_2$, respectively.*

- *Every proposition variable $X \in \mathcal{X}$ is a pre-clause where $Open(X) = \emptyset$.*

- *If P is a pre-clause then $\circ P$ is also a pre-clause where $Open(\circ P) = Open(P)$.*

- *If P is a pre-clause such that every free occurrence of the proposition variable X appears in the scope of at least one $\circ$ operator then $\nu X.P$ is also a pre-clause where $Open(\nu X.P) = Open(P)$.*

- *If P and Q are pre-clauses then $P \wedge Q$ is also a pre-clause where $Open(P \wedge Q) = Open(P) \cup Open(Q)$.*

- *If A is an antecedent and P is a pre-clause then $A \rightarrow P$ is a pre-clause where $Open(A \rightarrow P) = (FRV(A) \cup Open(P)) \setminus Bind(A)$.*

- *If P is a pre-clause which does not contain any free proposition variable, and $Open(P) = \emptyset$, then P is an* extended program clause.

Observe that according to this definition, propositional extended program clauses are not in general also first-order extended program clauses because, as was discussed above, negative literals have not been admitted as pre-clauses. This restriction is motivated by the following definition of (proper) program clauses; negative literals could have been allowed for extended program clauses.

However, these will in the sequel mainly be needed to construct formulas guaranteeing the compatibility of program clauses (via conflict schemes), and the more general definition of extended program clauses turns out to be unnecessary for this purpose.

The variable conditions imposed on extended program clauses are similar to those required of *allowed programs* for SLDNF resolution (cf. LLOYD [35]) in that bindings can only be established by positive literals. Additionally, it is required that every rigid subject variable occurring to the right of the implication sign is bound by an antecedent. This restriction helps to ensure that at every state only finitely many ground instances of program clauses can "fire".

A binding for a rigid variable x can only be established by some atomic formula $P(t_1, \ldots, t_m)$ if x appears as one of the terms t_i and all occurrences of x in $t_1, \ldots, t_m$ are unnested. For example, a binding for x cannot be established by a formula like $P(f(x))$: determining which values may be substituted for x in a given state σ_i to make this formula true, the interpreter would have to compute the set $f^{-1}(\sigma_i(P)) \subseteq |\mathcal{S}|$; besides, this set might be infinite, violating the requirement that every program clause have only finitely many applicable ground instances. Similarly, an equation $t_1 = t_2$ can only establish a binding for x if $t_1 \equiv x$ and x does not occur in t_2 (the variable conditions ensure that all rigid variables appearing in t have already been bound before). These restrictions enable a mechanical computation of the set of values bound by an antecedent, either by evaluating the term t in an equation $x = t$ or by a projection of the current extension of P to the argument position(s) where the bound variable x appears.

Examples for extended program clauses (where $x, y \in \mathcal{Y}$ are rigid variables, and $z \in \mathcal{Z}$ is a flexible variable) include

$$P(x) \vee Q(c_1, x) \rightarrow R(c_2, x) \wedge S(f(x)),$$

$$P(x) \rightarrow (Q(x,y) \rightarrow z = x + y \ \mathsf{atnext}\ \neg S(y))$$
$$\equiv P(x) \rightarrow (Q(x,y) \rightarrow \nu X.(\neg S(y) \rightarrow z = x + y) \wedge (S(y) \rightarrow \circ X)),$$

$$P(x) \rightarrow (y = 3 * x \wedge Q(y) \rightarrow 4 * z = x \ \mathsf{unless}\ R(x,y))$$
$$\equiv P(x) \rightarrow (y = 3 * x \wedge Q(y) \rightarrow (\nu X. \neg R(x,y) \rightarrow 4 * z = x \wedge \circ X)),$$

whereas

$$R(c_2, x) \wedge S(f(x)) \rightarrow P(x) \vee Q(c_1, x),$$
$$P(x) \rightarrow \circ Q(x,y), \qquad \text{or}$$
$$P(x) \wedge Q(x,y) \rightarrow z = x + y \ \mathsf{atnext}\ \neg S(y)$$

are not extended program clauses. (Observe that the last example fails because of $Bind(P(x) \wedge Q(x,y)) = \emptyset$. This somewhat unpleasant situation — in view of the semantically equivalent second positive example for extended program clauses — could be remedied, but the definition given above allows for a more uniform treatment of boolean connectives in antecedents and, in particular, for the construction of the conjunctive normal form of an antecedent without changing the set of bound variables.)

The additional restrictions placed on proper program clauses are similar as in the propositional case. The crucial point in the definition is again to rule out cyclic dependencies among module components. Technically, the definitions are somewhat more difficult because now even an atomic pre-clause may "read" flexible variables modelling module state components which have to be defined before the evaluation of the clause. Assume the set $\mathcal{Z}$ of flexible variables, similarly as the set $\mathcal{V}$ of proposition letters in chapter 4, to be partitioned into sets $\mathcal{Z}_E$ and $\mathcal{Z}_\Pi$ representing state components of the environment and the system, respectively, as well as the set $\mathcal{P}$ of predicate symbols to be partitioned into $\mathcal{P}_E$ and $\mathcal{P}_\Pi$ in an analogous way. Let $\mathcal{C} = \mathcal{Z} \cup \mathcal{P}$ denote the set of all state components, and $\mathcal{C}_E$ and $\mathcal{C}_\Pi$ denote the subsets of state components owned by the environment or the module, respectively. Again, the dependency order $<$ has to be given as a partial order on $\mathcal{C}_\Pi$. This order is extended to $\mathcal{C}$ by defining $\lambda < \mu$ for every environment component $\lambda \in \mathcal{C}_E \cup \{\mathsf{t}, \mathsf{f}\}$ and module component $\mu \in \mathcal{C}_\Pi$.

Moreover, all equations appearing in the consequent parts of pre-clauses are required to be of the form $z = t$ where $z \in \mathcal{Z}_\Pi$ is a flexible subject variable and t a term such that they can be interpreted operationally as assignments. The formal definition is as follows:

Definition 5.3 (program clauses)

- *Literals L and the set $Bind(L)$ of rigid subject variables which can be bound by L have been defined before. Additionally, the set $read(L)$ of state components needed for the evaluation of L in a state is defined by*

$$\begin{aligned} read(P(t_1,\ldots,t_m)) &= read(\neg P(t_1,\ldots,t_m)) = \{P\} \cup \bigcup_{i=1}^{m} FFV(t_i), \\ read(t_1 = t_2) &= read(t_1 \neq t_2) = FFV(t_1) \cup FFV(t_2). \end{aligned}$$

- *Every literal is an* antecedent*. If A and B are antecedents, then $A \wedge B$ and $A \vee B$ also are antecedents where $Bind(A \wedge B) = Bind(A \vee B) = Bind(A) \cap Bind(B)$ and $read(A \wedge B) = read(A \vee B) = read(A) \cup read(B)$.*

- *Atomic formulas of the form $P(t_1,\ldots,t_m)$ where $P \in \mathcal{P}_\Pi$ or $z = t$ where $z \in Z_\Pi$ are pre-clauses if $z' < P$ (resp., $z' < z$) holds for every flexible variable $z' \in \bigcup_{i=1}^m FFV(t_i)$ ($z' \in FFV(t)$, respectively). The sets*

Open$(P(t_1, \ldots, t_m))$ *and Open*$(z = t)$ *are defined as the sets of free rigid subject variables appearing in* $P(t_1, \ldots, t_m)$ *and* $z = t$*, respectively. The set of state components "written" to in the evaluation state of the pre-clause is defined as*

$$write(P(t_1, \ldots, t_n)) = \{P\}, \qquad write(z = t) = \{z\}.$$

- *If* P *is a pre-clause then* $\circ P$ *is also a pre-clause where* $Open(\circ P) = Open(P)$ *and* $write(\circ P) = \emptyset$*. Moreover, if* $X \in \mathcal{X}$ *is a proposition variable then* $\circ X$ *is a pre-clause where* $Open(\circ X) = \emptyset$ *and* $write(\circ X) = \emptyset$*.*
- *If* P *is a pre-clause and* X *is a proposition variable then* $\nu X.P$ *is also a pre-clause where* $Open(\nu X.P) = Open(P)$ *and* $write(\nu X.P) = write(P)$*.*
- *If* P *and* Q *are pre-clauses then* $P \wedge Q$ *is also a pre-clause where* $Open(P \wedge Q) = Open(P) \cup Open(Q)$ *and* $write(P \wedge Q) = write(P) \cup write(Q)$*.*
- *If* A *is an antecedent and* P *is a pre-clause such that* $\kappa < \lambda$ *holds for all state components* $\kappa \in read(A)$*,* $\lambda \in write(P)$ *then* $A \rightarrow P$ *is a pre-clause where* $Open(A \rightarrow P) = (FRV(A) \cup Open(P)) \setminus Bind(A)$ *and* $write(A \rightarrow P) = write(P)$*.*
- *If* P *is a pre-clause which does not contain any free proposition variable and where* $Open(P) = \emptyset$*, then* P *is a* program clause.

Of the examples given for extended program clauses, all but the last one can also be turned into examples for proper program clauses if a suitable order is defined on the state components. For example, for the second clause it is required that $P < z$, $Q < z$ and $S < z$ hold. The last example fails to be a program clause because the equation $4 * z = x$ is not of the required form (replacing it by $z = x/4$ would yield an acceptable program clause).

Notice that by theorem 2.30, the universal closure of every extended program clause expresses a safety property.

5.2 Transition schemes

In a similar way as propositional extended program clauses were represented by transition graphs which guided the interpreter algorithm and allowed the formulation of the compatibility condition for propositional program clauses, an automata-theoretic framework for the representation of extended first-order program clauses will be defined in this section. The node labels of these *transition schemes* not only consist of a set of literals, but also record the set of rigid subject variables which may appear in these literals (i.e. which have already been bound by some antecedent). In this way, the variable conditions for (extended) program clauses stated in the definitions 5.1 and 5.2 can be represented in transition schemes.

Definition 5.4 (transition schemes)

A transition scheme $T = (N, V, I, S, \vartheta, \zeta, n_0)$ *consists of:*

- *a finite non-empty set* N *of* nodes*,*
- *a mapping* V *assigning every node* n *a finite set of rigid subject variables* $V(n) \subseteq \mathcal{Y}$ *which are* bound at node n*,*
- *an acyclic relation* $I \subseteq N \times N$ *representing* internal edges*, i.e. for no* $n \in N$ *it holds that* $(n, n) \in I^+$*,*
- *a relation* $S \subseteq N \times N$ *representing* state transitions *(or* steps*),*
- *a* node labelling ϑ *assigning a finite set* $\vartheta(n)$ *of atomic formulas to every node* $n \in N$ *such that all rigid subject variables appearing in formulas of* $\vartheta(n)$ *are contained in the set* $V(n)$ *of variables which are bound at node* n*,*
- *an* edge labelling ζ *assigning an antecedent* $\zeta(n, n')$ *defined as in definition 5.1 to every internal edge* $(n, n') \in I$ *such that for every* $(n, n') \in I$*,*

  $$FRV(\zeta(n, n')) \subseteq V(n) \cup \mathit{Bind}(\zeta(n, n')),$$

  i.e. every rigid subject variable appearing in $\zeta(n, n')$ *either is among the bound variables* $V(n)$ *of node* n*, or it is bound by the antecedent* $\zeta(n, n')$ *itself,*
- *and a designated initial node* $n_0 \in N$ *where* $V(n_0) = \emptyset$*,*

where the following variable conditions hold for every two nodes $n, n' \in N$*:*

$$\begin{array}{lcll} V(n') & = & V(n) & \text{if } (n, n') \in S \quad \text{and} \\ V(n') & = & V(n) \cup \mathit{Bind}(\zeta(n, n')) & \text{if } (n, n') \in I. \end{array}$$

Again, the node labels $\vartheta(n)$ which are defined as finite sets of atomic formulas will frequently be identified with the conjunction of their members. Moreover, the finite set $V(n) = \{y_1, \ldots, y_k\}$ of rigid subject variables bound at a node n will sometimes be identified with the k-tuple $(y_1, \ldots, y_k)$ of variables (assuming a fixed numbering of the rigid subject variables).

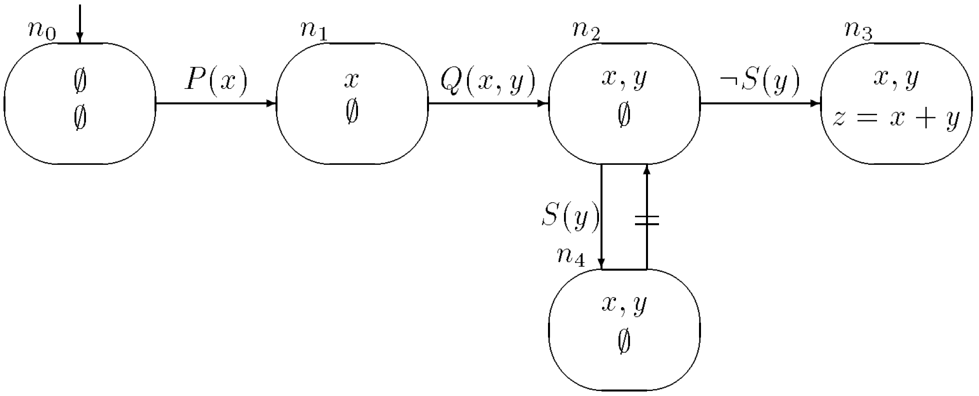


Figure 5.1: Graphical representation of a transition scheme

Graphical representations of transition schemes can be given similarly as those of (propositional) transition graphs. Every node n is inscribed by both the variables bound at the respective node (represented by the set $V(n)$) and the formulas contained in its node label $\vartheta(n)$; see figure 5.1 for an example.

In analogy to propositional transition graphs, the following restricted forms of transition schemes can be defined:

Definition 5.5 (regular transition schemes)
A transition scheme $T = (N, V, I, S, \vartheta, \zeta, n_0)$ is called regular *if for every node $n \in N$ it holds that $I(n) = \emptyset$ or $S(n) = \emptyset$. A node n of a regular transition scheme T satisfying $S(n) = \emptyset$ is called a* state *of T.*

Definition 5.6 (normal transition schemes)
A transition scheme $T = (N, V, I, S, \vartheta, \zeta, n_0)$ is called normal *if for every internal edge $(n, n') \in I$, the edge label $\zeta(n, n')$ is a non-empty finite disjunction $L_1 \vee \ldots \vee L_k$ of literals.*

Definition 5.7 (monotonic transition schemes)
A transition scheme $T = (N, V, I, S, \vartheta, \zeta, n_0)$ is called monotonic *if for all nodes $n, n' \in N$ and terms t_1, t_2 the following condition holds:*

$$(n, n') \in I, t_1 = t_2 \in \vartheta(n) \quad \Longrightarrow \quad t_1 = t_2 \in \vartheta(n').$$

Notice that this definition is somewhat less restrictive than the straightforward generalization of the corresponding definition 4.6 for transition graphs in that only equations have to be copied into the node labels of I-successors. This restriction suffices here because only equations can lead to inconsistencies between two given node labels.

Definition 5.8 (complete transition schemes)
A transition scheme $T = (N, V, I, S, \vartheta, \zeta, n_0)$ is called complete *if for every node $n \in N$ where $I(n) \neq \emptyset$, the universal closure of the disjunction of all labels of internal edges starting in n, i.e. the formula $(\forall *) \bigvee_{(n,n') \in I} \zeta(n, n')$ is (first-order) valid.*

For example, the transition scheme of figure 5.1 is regular, normal and monotonic, but not complete.

Again, transition schemes have to be related with temporal structures and, in particular, the notion of acceptance of a temporal structure by a transition scheme has to be defined. To this end, markings of transition schemes over temporal structures are introduced where in contrast to the propositional case, not only the "active" nodes have to be recorded for every state of the temporal structure, but also the bindings of object tuples to variables bound in the respective nodes. Formally, the definition is as follows:

Definition 5.9 (marking of a transition scheme, acceptance)
Let $T = (N, V, I, S, \vartheta, \zeta, n_0)$ be a transition scheme and $\mathcal{K} = (\mathcal{S}, \sigma)$ be a temporal structure. The marking *of T over $\mathcal{K}$ is defined as the sequence $M^{\mathcal{K}} = (m_0^{\mathcal{K}}, m_1^{\mathcal{K}}, \ldots)$ where for every $i \geq 0$ and every $n \in N$, $m_i^{\mathcal{K}}(n) \subseteq |\mathcal{S}|^{|V(n)|}$ is the smallest set of object tuples (indexed by the variables in $V(n)$) satisfying the following conditions:*

- $m_0^{\mathcal{K}}(n_0) = \{()\}$ *(where $()$ denotes the empty tuple)*
- *If $n, n' \in N$ are nodes where $V(n) = (y_1, \ldots, y_k)$ and $V(n') = (y'_1, \ldots, y'_l)$ (using the tuple interpretation of $V(n)$), $(n, n') \in I$ is an internal edge and there is some valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ of the rigid subject variables such that $(\eta(y_1), \ldots, \eta(y_k)) \in m_i^{\mathcal{K}}(n)$ and $\mathcal{K}_i^{\eta} \models \zeta(n, n')$ then $(\eta(y'_1), \ldots, \eta(y'_l)) \in m_i^{\mathcal{K}}(n')$.*
- *If $n, n' \in N$ are nodes such that $(n, n') \in S$ and $V(n) = V(n') = (y_1, \ldots, y_k)$ then $m_i^{\mathcal{K}}(n) \subseteq m_{i+1}^{\mathcal{K}}(n')$.*

$\mathcal{K}$ is accepted *by T if for every $i \geq 0$, every $n \in N$ where $V(n) = (y_1, \ldots, y_k)$ and all valuations $\eta : \mathcal{Y} \to |\mathcal{S}|$ such that $(\eta(y_1), \ldots, \eta(y_k)) \in m_i^{\mathcal{K}}(n)$, it holds that*

$$\mathcal{K}_i^{\eta} \models \vartheta(n).$$

In the sequel, for $n \in N$, $\eta(V(n)) \in m_i^{\mathcal{K}}(n)$ abbreviates the condition $(\eta(y_1), \ldots, \eta(y_k)) \in m_i^{\mathcal{K}}(n)$ for $V(n) = (y_1, \ldots, y_k)$.

A transition scheme T thus defines a property (i.e. a set of temporal structures), viz. the set of structures accepted by T. As was the case for extended program clauses and transition graphs in the propositional case discussed in chapter 4, the class of properties definable by transition schemes and extended program clauses (or, rather, their universal closures) coincide. Again, effective algorithms will be given to construct a transition scheme from an extended program clause and vice versa.

Algorithm 5.10 (associated transition scheme)

Let φ be an extended program clause. Then the transition scheme T^φ obtained by the following construction is called the *associated transition scheme* for φ.

Similarly as in algorithm 4.9, $T^\varphi = (N, V, I, S, \vartheta, \zeta, n_0)$ is obtained from a generalized transition scheme $\tilde{T} = (N, V, I, S, \tilde{\vartheta}, \zeta, n_0)$ whose node labelling $\tilde{\vartheta}$ assigns a set of pre-clauses to every node by restricting $\vartheta(n)$, for every $n \in N$, to atomic formulas.

It will again be assumed that all proposition variables X which appear bound in different subformulas $\nu X.F$ of φ are pairwise different which can be achieved by a suitable renaming of the bound propositional variables. For any such X, F_X denotes the "defining formula" for X in φ, i.e. the (by assumption unique) formula such that $\nu X.F_X$ appears as a subformula of φ. Since φ by definition is a closed νTL formula, such a formula F_X has to exist.

For every pre-clause P appearing as a subformula of φ, its *closure set* $cl(P)$ is defined in analogy to the propositional case by

$$\begin{array}{ll} cl(L) = \{L\} \quad \text{if } L \equiv P(t_1, \ldots, t_n) \text{ or } L \equiv t_1 = t_2 \text{ is a (positive) literal,} & \\ cl(X) = \{X\} \cup cl(F_X) \quad \text{for a proposition variable } X, & \\ cl(\circ P) = \{\circ P\}, & cl(P \wedge Q) = \{P \wedge Q\} \cup cl(P) \cup cl(Q), \\ cl(A \to P) = \{A \to P\}, & cl(\nu X.F_X) = \{\nu X.F_X\} \cup cl(F_X). \end{array}$$

Observe that, for every (sub-)pre-clause P of φ, $cl(P)$ is a finite set of such pre-clauses where $P \in cl(P)$ and that $\mathcal{K}^{\eta,\xi}(P) \subseteq \mathcal{K}^{\eta,\xi}(F)$ holds for every pre-clause $F \in cl(P)$, every temporal structure $\mathcal{K}$, every valuation η of the rigid subject variables and every valuation ξ of the proposition variables satisfying $\xi(X) = \mathcal{K}^{\eta,\xi}(\nu X.F_X)$ for every proposition variable X appearing in φ.

Construct $\tilde{T}$, as in algorithm 4.9, starting from the initial node n_0 where $V(n_0) = \emptyset$ and $\tilde{\vartheta}(n_0) = cl(\varphi)$, and generating new nodes and edges according to the following rules. Whenever these say: "Construct a node n' where

$V(n') = V$ and $\tilde{\vartheta}(n') = F$ and an edge $(n, n') \in I$ labelled by $\zeta(n, n') = A$ (or an edge $(n, n') \in S$)" but some node n'' such that $V(n'') = V$ and $\tilde{\vartheta}(n'') = F$ has already been created, then only the edge (n, n'') with the proper label has to be inserted. If such an edge (of the required type) also exists, but A does not appear as a disjunct in its label $\zeta(n, n'') \equiv B$, then the label is changed to $B \vee A$.

1. For every node n whose label $\tilde{\vartheta}(n)$ contains a formula $A \rightarrow P$ create a node n' where $V(n') = V(n) \cup \mathit{Bind}(A)$ which is labelled by $\tilde{\vartheta}(n') = \mathit{cl}(P)$ and an internal edge $(n, n') \in I$ labelled by $\zeta(n, n') = A$.

2. For every node n whose label $\tilde{\vartheta}(n)$ contains a formula $\circ P$ create a node n' where $V(n') = V(n)$ which is labelled by $\tilde{\vartheta}(n') = \bigcup\{\mathit{cl}(P) \mid \circ P \in \tilde{\vartheta}(n)\}$ and a state transition $(n, n') \in S$.

Every node n created by this construction is labelled by a set of pre-clauses occurring as subformulas of φ, moreover, $V(n) \subseteq FRV(\varphi)$. Thus, only a finite number of different nodes can be created, establishing the termination of the construction. It has to be shown that T^{φ} qualifies as a transition scheme:

- I is acyclic: The same argument used in algorithm 4.9 applies to the above construction.

- $\bigcup\limits_{P \in \vartheta(n)} FRV(P) \subseteq V(n)$: This follows from the more general assertion

$$(*) \bigcup_{P \in \tilde{\vartheta}(n)} \mathit{Open}(P) \subseteq V(n)$$

  since $\mathit{Open}(P) = FRV(P)$ for atomic pre-clauses P. The assertion $(*)$ is established by an induction on the above algorithm:

  - $n = n_0$: By definition, $\mathit{Open}(\varphi) = \emptyset$, and since $X \notin \mathit{cl}(\varphi)$ for every proposition variable X (because φ is closed, and every proposition variable occurring in φ therefore has to appear in the scope of at least one $\circ$ operator) it follows that $\bigcup_{P \in \mathit{cl}(\varphi)} \mathit{Open}(P) = \mathit{Open}(\varphi)$, hence $\bigcup_{P \in \tilde{\vartheta}(n_0)} \mathit{Open}(P) = \bigcup_{P \in \mathit{cl}(\varphi)} \mathit{Open}(P) = \emptyset = V(n_0)$.
  - n has been introduced by an application of rule 1 above: Then there is some node n' whose label $\tilde{\vartheta}(n')$ contains some pre-clause $A \rightarrow P$, and by the induction hypothesis, $\mathit{Open}(A \rightarrow P) \subseteq V(n')$, i.e. $\mathit{Open}(P) \subseteq V(n') \cup \mathit{Bind}(A) = V(n)$. Let $X \in \mathit{cl}(P)$ be any proposition variable appearing in the closure of P. Then, by the construction of $\tilde{T}$ there is some $(I \cup S)^*$-predecessor n'' of n' such

that $\nu X.F_X \in \tilde{\vartheta}(n'')$, hence $F_X \in \tilde{\vartheta}(n'')$ from which it follows by induction hypothesis that $Open(F_X) \subseteq V(n'')$. Now, since $(n'', n') \in (I \cup S)^*$, the construction of $\tilde{T}$ ensures that $V(n'') \subseteq V(n') \subseteq V(n)$. For every formula $F \in cl(P)$ it holds that either F is a subformula of P and $Open(F) \subseteq Open(P)$, or that F is a subformula of some formula F_X such that $X \in cl(P)$ and $Open(F) \subseteq Open(F_X)$ (since formulas $A \to F$ are not decomposed by the closure operation). Thus, in both cases, $Open(F) \subseteq V(n)$ for every $F \in cl(P)$, proving $(*)$.

  – n has been introduced by an application of rule 2 above: Then there is some node n' such that $\tilde{\vartheta}(n) = \bigcup\{cl(P) \mid \circ P \in \tilde{\vartheta}(n')\}$, and by the induction hypothesis it follows that $\bigcup\{Open(\circ P) \mid \circ P \in \tilde{\vartheta}(n')\} \subseteq V(n')$, hence by $Open(\circ P) = Open(P)$ and $V(n) = V(n')$ it holds that $Open(P) \subseteq V(n')$ for every P such that $\circ P \in \tilde{\vartheta}(n')$. Applying the same arguments as in the preceding case yields the assertion $(*)$.

- $FRV(\zeta(n, n')) \subseteq V(n) \cup Bind(\zeta(n, n'))$: Let $(n, n') \in I$ be any internal edge. Then this edge has been created by, say, k applications of rule 1 applied to node n, i.e. there are formulas $A_j \to P_j$ (where $1 \leq j \leq k$) such that $\zeta(n, n') = A_1 \vee \ldots \vee A_k$ and $\tilde{\vartheta}(n') = cl(P_1) = \ldots = cl(P_k)$. By the assertion $(*)$ proven above it follows that $Open(A_j \to P_j) \subseteq V(n)$, implying $FRV(A_j) \subseteq V(n) \cup Bind(A_j)$ by the definition of $Open(A_j \to P_j)$. Moreover, $V(n') = V(n) \cup Bind(A_1) = \ldots = V(n) \cup Bind(A_k)$, hence

$$(**) \quad V(n) \cup \bigcup_{j=1}^{k} Bind(A_j) = V(n) \cup \bigcap_{j=1}^{k} Bind(A_j).$$

  Together, this implies

$$\begin{aligned} FRV(\zeta(n, n')) &= \bigcup_{j=1}^{k} FRV(A_j) \\ &\subseteq V(n) \cup \bigcup_{j=1}^{k} Bind(A_j) \\ &= V(n) \cup \bigcap_{j=1}^{k} Bind(A_j) \\ &= V(n) \cup Bind(\zeta(n, n')), \end{aligned}$$

  which was to be shown.

- $V(n') = V(n) \cup Bind(\zeta(n, n'))$ for $(n, n') \in I$: follows immediately from rule 1 which introduces all internal edges and assertion $(**)$ above.

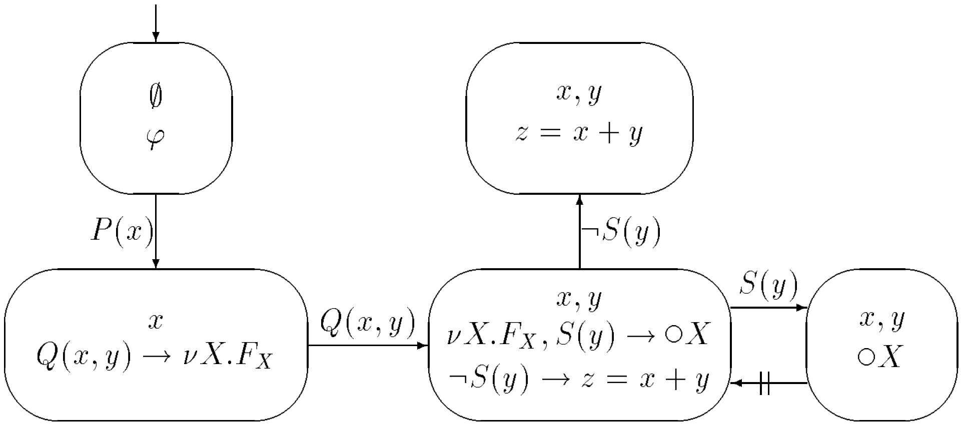


Figure 5.2: A generalized transition scheme (example 5.11)

- $V(n') = V(n)$ for $(n, n') \in S$: follows directly from rule 2 which introduces all state transitions of $\tilde{T}$.

♠

Example 5.11 Figure 5.2 represents the generalized transition scheme $\tilde{T}$ which is constructed by algorithm 5.10 for the extended program clause

$$
\begin{aligned}
\varphi &\equiv P(x) \to (Q(x, y) \to z = x + y \ \mathsf{atnext}\ \neg S(y)) \\
&\equiv P(x) \to (Q(x, y) \to \nu X.(\neg S(y) \to z = x + y) \wedge (S(y) \to \circ X))
\end{aligned}
$$

The transition scheme of figure 5.1 results from $\tilde{T}$ by restricting the node labels to literals. It accepts every temporal structure which is a model of the universal closure of φ. The following lemma shows that this correspondence holds for arbitrary extended program clauses.

Lemma 5.12 *Let φ be an extended program clause. Then the associated transition scheme T^φ accepts exactly the models of the universal closure $(\forall *)\varphi$ of φ.*

Proof: The proof proceeds similarly as in lemma 4.11. For the "if" direction, assume $\mathcal{K} = (\mathcal{S}, \sigma)$ is a model of $(\forall *)\varphi$ and let $M = (m_0, m_1, \ldots)$ denote the marking of T^φ over $\mathcal{K}$ as defined in definition 5.9. Again, the definition of M is independent of the node labelling ϑ of T^φ, hence M is also the marking of the generalized transition scheme $\tilde{T}$ generated by algorithm 5.10. Define a partial order $<$ on the proposition variables appearing in φ by $X < Y$ iff $\nu Y.F_Y$ is a subformula of $\nu X.F_X$ and define, for every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ of the rigid subject variables, a valuation ξ of these proposition variables by induction on $<$ as follows:

$$\xi(X) = \mathcal{K}^{\eta,\xi_X}(\nu X.F_X) \quad \text{where } \xi_X(Y) = \left\{ \begin{array}{ll} \xi(Y) & \text{if } Y < X \\ \emptyset & \text{else} \end{array} \right\}.$$

Again, it holds for every proposition variable X appearing in φ that $\xi(X) = \mathcal{K}^{\eta,\xi}(F_X)$, and that, for every pre-clause P which appears as a subformula of φ,

$$(H) \quad \mathcal{K}^{\eta,\xi}(P) = \bigcap_{Q \in cl(P)} \mathcal{K}^{\eta,\xi}(Q).$$

Let us now prove

$$(*) \quad \mathcal{K}_i^{\eta,\xi} \models \tilde{\vartheta}(n)$$

for every $i \in I\!N$, every $n \in N$ and every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ such that $\eta(V(n)) \in m_i(n)$ (and the propositional valuation ξ defined above). In particular, $(*)$ implies that $\mathcal{K}_i^\eta \models \vartheta(n)$ for all $i \in I\!N$ and $n \in N$ if $\eta(V(n)) \in m_i(n)$ (because the node labels of T^φ do not contain propositional variables), i.e. T^φ accepts $\mathcal{K}$.

The proof of $(*)$ runs by induction on the definition of M:

Induction Base: $i = 0, n = n_0$. Then $\tilde{\vartheta}(n_0) = cl(\varphi)$ and the assertion follows from $\mathcal{K}_0^{\eta,\xi} \models \varphi$ for all valuations η and ξ and the assertion (H).

Induction Step 1: There is some $n' \in N$ where $(n', n) \in I$, $\mathcal{K}_i^\eta \models \zeta(n', n)$ and $\eta(V(n')) \in m_i(n')$: Then n has been introduced by (possibly multiple) applications of rule 1 to node n', hence there are $k \geq 1$ formulas $A_1 \to P_1, \ldots, A_k \to P_k \in \tilde{\vartheta}(n')$ such that $\zeta(n', n) = A_1 \vee \ldots \vee A_k$ and $\tilde{\vartheta}(n) = cl(P_1) = \ldots = cl(P_k)$ and, for some $1 \leq j \leq k$, $\mathcal{K}_i^\eta \models A_j$. By the induction hypothesis, $\mathcal{K}_i^{\eta,\xi} \models A_j \to P_j$, yielding $\mathcal{K}_i^{\eta,\xi} \models P_j$ and, by (H), $\mathcal{K}_i^{\eta,\xi} \models cl(P_j) = \tilde{\vartheta}(n)$.

Induction Step 2: $i > 0$ and there is some $n' \in N_{i-1}$ where $(n', n) \in S$ and $\eta(V(n')) \in m_{i-1}(n')$: Then the induction hypothesis yields $\mathcal{K}_{i-1}^{\eta,\xi} \models \circ P$ for every pre-clause $\circ P \in \tilde{\vartheta}(n')$, i.e. $\mathcal{K}_i^{\eta,\xi} \models P$ and by (H), $\mathcal{K}_i^{\eta,\xi} \models cl(P)$ for every such P from which the assertion $(*)$ follows by the definition of $\tilde{\vartheta}(n)$.

By the definition of M, for every valuation η satisfying $\eta(V(n)) \in m_i(n)$ one of the cases above has to hold.

Conversely, assume that the temporal structure $\mathcal{K}$ is accepted by T^φ. Again, let $M = (m_0, m_1, \ldots)$ denote the marking of $\tilde{T}$ over $\mathcal{K}$. For a given valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ of the rigid subject variables, define the propositional valuation ξ by $\xi(X) = \{i \in I\!N \mid \exists n \in N \; (\eta(V(n)) \in m_i(n)$ and $X \in \tilde{\vartheta}(n)$ or $\nu X.F_X \in$

$\tilde{\vartheta}(n))\}$. For every pre-clause F, every $i \geq 0$, every $n \in N$ and every valuation η such that $\eta(V(n)) \in m_i(n)$ we prove that

$$(*) \quad \mathcal{K}_i^{\eta,\xi} \models F \quad \text{if } F \in \tilde{\vartheta}(n)$$

In particular, then, $\mathcal{K}_0^{\eta,\xi} \models \varphi$ for every valuation η since $\varphi \in \tilde{\vartheta}(n_0)$ and the condition on η is trivial, implying $\mathcal{K} \models (\forall *)\varphi$ because φ is a closed formula. The proof of $(*)$ runs by induction on the pre-clause F:

- $F \equiv P(t_1, \ldots, t_m)$ or $F \equiv t_1 = t_2$: Then $F \in \tilde{\vartheta}(n)$ implies $F \in \vartheta(n)$ (where ϑ denotes the node labelling of T^{φ}), hence $\mathcal{K}_i^{\eta} \models F$ holds by the acceptance of $\mathcal{K}$ by T^{φ}.

- $F \equiv X$: Then $\mathcal{K}^{\eta,\xi}(X) = \xi(X)$ and by the assumptions $X \in \tilde{\vartheta}(n)$, $\eta(V(n)) \in m_i(n)$, and the definition of ξ, the assertion follows.

- $F \equiv A \rightarrow P$: Then $\tilde{T}$ contains a node n' such that $(n, n') \in I$, $\tilde{\vartheta}(n') = cl(P)$ (in particular, $P \in \tilde{\vartheta}(n')$), A appears as a disjunct in $\zeta(n, n')$ and $V(n') = V(n) \cup Bind(A)$ (cf. the proof of assertion $(**)$ in algorithm 5.10). If $\mathcal{K}_i^{\eta,\xi} \models A$, the assumption $\eta(V(n)) \in m_i(n)$ implies $\eta(V(n')) \in m_i(n')$, hence by the induction hypothesis, $\mathcal{K}_i^{\eta,\xi} \models P$ which proves $\mathcal{K}_i^{\eta,\xi} \models A \rightarrow P$.

- $F \equiv P \wedge Q$: By construction of $\tilde{T}$, it follows that both $P \in \tilde{\vartheta}(n)$ and $Q \in \tilde{\vartheta}(n)$ from which the assertion follows by the induction hypothesis.

- $F \equiv \circ P$: Then $\tilde{T}$ contains some node n' such that $(n, n') \in S$, $P \in \tilde{\vartheta}(n')$ and $V(n') = V(n)$. Moreover, by $\eta(V(n)) \in m_i(n)$ and $m_i(n) \subseteq m_{i+1}(n')$ it follows that $\eta(V(n')) \in m_{i+1}(n')$ which by induction hypothesis implies $\mathcal{K}_{i+1}^{\eta,\xi} \models P$, i.e. $\mathcal{K}_i^{\eta,\xi} \models \circ P$.

- $F \equiv \nu X.F_X$: Assume more generally that $j \geq 0$ such that $j \in \xi(X)$. Then there is some $n \in N$ such that $\eta(V(n)) \in m_j(n)$ and $X \in \tilde{\vartheta}(n)$ or $\nu X.F_X \in \tilde{\vartheta}(n)$. In both cases, this implies $F_X \in \tilde{\vartheta}(n)$ by the construction of $\tilde{T}$. Hence, by the induction hypothesis, $\mathcal{K}_j^{\eta,\xi} \models F_X$ and, since $j \in \xi(X)$ was chosen arbitrarily, it follows that $\xi(X) \subseteq \mathcal{K}^{\eta,\xi}(F_X)$, hence also $\xi(X) \subseteq \mathcal{K}^{\eta,\xi}(\nu X.F_X)$. By the assumptions $\nu X.F_X \in \tilde{\vartheta}(n)$ and $\eta(V(n)) \in m_i(n)$, it follows that $i \in \xi(X)$, yielding $i \in \mathcal{K}^{\eta,\xi}(\nu X.F_X)$ as required.

♠

Again, if φ is a proper program clause, the associated transition scheme satisfies additional properties regarding the dependency order $<$ defined on state components:

Lemma 5.13 *Let φ be a program clause and T^φ its associated transition scheme. Let $\pi = (n^0, n^1, \ldots, n^k)$ be an I^*-path in T^φ, and let $\kappa \in \mathcal{C}$ be a state component such that there is some formula $A \in \vartheta(n^k)$ and $\kappa \in write(A)$. Then $\lambda < \kappa$ holds for every state component $\lambda \in \mathcal{C}$ appearing in some edge label $\zeta(n^0, n^1), \ldots, \zeta(n^{k-1}, n^k)$ along the path π and every flexible variable $\lambda \in FFV(B)$ appearing in some formula $B \in \vartheta(n^j)$ for $0 \leq j \leq k$.*

Proof: analogous to the proof of lemma 4.12 using the variable conditions on proper program clauses. ♠

Again, all state components $\kappa \in write(A)$ "written" by some formula $A \in \vartheta(n)$ appearing in the node label of the transition scheme associated with a proper program clause have to be module state components $\kappa \in \mathcal{C}_\Pi$ by the corresponding restriction on the formation of pre-clauses for proper program clauses.

In a similar way as extended propositional program clauses could be constructed from transition graphs, the following algorithm generates an extended (first-order) program clause for every transition scheme:

Algorithm 5.14 (associated extended program clause)
Let $T = (N, V, I, S, \vartheta, \zeta, n_0)$ be a transition scheme where $N = \{n_0, \ldots, n_k\}$ is the set of nodes of T. The *associated extended program clause* φ^T is defined as follows using the proposition variables $X_0, \ldots, X_k$:

For $0 \leq j \leq k$ and every subset $X \subseteq \{X_0, \ldots, X_k\}$ of the proposition variables, define the formulas ψ_j^X and δ_j^X by

$$\begin{aligned} \psi_j^X &\equiv \vartheta(n_j) \wedge \bigwedge_{(n_j,n_l)\in I} (\zeta(n_j,n_l) \rightarrow \delta_l^X) \wedge \bigwedge_{(n_j,n_l)\in S} \circ\delta_l^X, \\ \delta_j^X &\equiv \left\{ \begin{array}{ll} X_j & \text{if } X_j \in X \\ \nu X_j.\psi_j^{X\cup\{X_j\}} & \text{otherwise} \end{array} \right\}. \end{aligned}$$

Now, define $\varphi^T :\equiv \delta_0^\emptyset \equiv \nu X_0.\psi_0^{\{X_0\}}$.

By induction on $\{X_0, \ldots, X_k\} \setminus X$, we show that all ψ_j^X are pre-clauses whose free proposition variables are among the variables in X and where $Open(\psi_j^X) \subseteq V(n_j)$. In particular, φ^T is an extended program clause.

First, $\vartheta(n_j)$ only contains atomic formulas $P \equiv P(t_1, \ldots, t_m)$ or $P \equiv t_1 = t_2$ (and, in particular, every such P is a pre-clause) such that $Open(P) = FRV(P) \subseteq V(n_j)$, hence the assertion holds for the first conjunct of ψ_j^X.

Consider any conjunct $\zeta(n_j, n_l) \rightarrow \delta_l^X$ corresponding to some internal edge $(n_j, n_l) \in I$. Then $\zeta(n_j, n_l)$ is an antecedent, and there are two cases:

$\underline{X_l \in X:}$ Then $\delta_l^X \equiv X_l$ which is a pre-clause, and $Open(\zeta(n_j, n_l) \to \delta_l^X) \subseteq Open(X_l) = \emptyset$, yielding the assertion for this conjunct.

$\underline{X_l \notin X:}$ Then $\delta_l^X \equiv \nu X_l.\psi_l^{X \cup \{X_l\}}$. By the induction hypothesis, $\psi_l^{X \cup \{X_l\}}$ is a pre-clause and by the acyclicity of I and the construction of $\psi_l^{X \cup \{X_l\}}$, every free occurrence of X_l has to be in the scope of at least one $\circ$ operator, hence δ_l^X also is a pre-clause. Moreover, by the induction hypothesis, all free proposition variables occurring in $\psi_l^{X \cup \{X_l\}}$ are among $X \cup \{X_l\}$, hence the free proposition variables of δ_l^X are among those in X, and $Open(\nu X_l.\psi_l^{X \cup \{X_l\}}) = Open(\psi_l^{X \cup \{X_l\}}) \subseteq V(n_l)$, and by $V(n_l) = V(n_j) \cup Bind(\zeta(n_j, n_l))$ and $FRV(\zeta(n_j, n_l)) \subseteq V(n_l)$ it follows that $Open(\zeta(n_j, n_l) \to \delta_l^X) = (FRV(\zeta(n_j, n_l)) \cup Open(\delta_l^X)) \setminus Bind(\zeta(n_j, n_l)) \subseteq V(n_j)$ as was to be shown.

The assertion for the conjuncts $\circ\delta_l^X$ corresponding to the state transitions $(n_j, n_l) \in S$ is proven similarly. ♠

Lemma 5.15 *Let T be a transition scheme and φ^T be its associated extended program clause. Then T accepts exactly the models of the universal closure $(\forall *)\varphi^T$ of φ^T.*

Proof: Let $\mathcal{K} = (\mathcal{S}, \sigma)$ be a model of $(\forall *)\varphi^T$ and let $M = (m_0, m_1, \ldots)$ denote the marking of T over $\mathcal{K}$. It has to be shown for every $i \geq 0$ that $\mathcal{K}_i^\eta \models \vartheta(n)$ for every $n \in N$ and every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ such that $\eta(V(n)) \in m_i(n)$. By induction on the definition of M, we show the more general assertion that for every $i \geq 0$, every node $n_j \in N$ and every valuation η satisfying $\eta(V(n_j)) \in m_i(n_j)$ there is some set $X \subseteq \{X_0, \ldots, X_k\}$ of proposition variables such that

$$(*) \quad \mathcal{K}_i^{\eta, \xi_X} \models \delta_j^X$$

where ξ_X is a valuation of the proposition variables satisfying $\xi(X_m) = \mathcal{K}^{\eta,\xi}(\nu X_m.\psi_m^X)$ for every $X_m \in X$. As a consequence, either $\mathcal{K}_i^{\eta,\xi_X} \models X_j$ which by the condition on ξ_X implies $\mathcal{K}_i^{\eta,\xi_X} \models \psi_j^X$, or $\mathcal{K}_i^{\eta,\xi_X} \models \nu X_j.\psi_j^{X \cup \{X_j\}}$. In both cases, it follows that $\mathcal{K}_i^\eta \models \vartheta(n_j)$, implying the acceptance of $\mathcal{K}$ by T.

Induction Base: $i = 0$, $j = 0$. Since $\mathcal{K} \models \varphi^T \equiv \delta_0^\emptyset$ by assumption, the assertion holds for $X = \emptyset$ (and an arbitrary propositional valuation $\xi_\emptyset$).

Induction Step 1: $n_j \in N$ such that $(n_j, n_l) \in I$, $\eta(V(n_j)) \in m_i(n_j)$, and $\mathcal{K}_i^\eta \models \zeta(n_j, n_l)$. By the induction hypothesis, $\mathcal{K}_i^{\eta,\xi_X} \models \delta_j^X$ for some $X \subseteq \{X_0, \ldots, X_k\}$. If $X_j \in X$, $\delta_j^X \equiv X_j$, hence $i \in \xi_X(X_j)$ and by $\xi_X(X_j) = \mathcal{K}^{\eta,\xi_X}(\nu X_j.\psi_j^X)$, it follows that $\mathcal{K}_i^{\eta,\xi_X} \models \psi_j^X$. By definition of ψ_j^X and using the assumption $\mathcal{K}_i^\eta \models \zeta(n_j, n_l)$, this implies $\mathcal{K}_i^{\eta,\xi_X} \models \delta_l^X$.

If $X_j \notin X$, $\mathcal{K}_i^{\eta,\xi_X} \models \delta_j^X \equiv \nu X_j.\psi_j^{X \cup \{X_j\}}$. Let $J := \mathcal{K}^{\eta,\xi_X}(\delta_j^X)$ and $\xi' := \xi[J/X_j]$. Since $X_j \notin X$ does not occur in δ_j^X as a free proposition variable, $\xi'(X_j) = \mathcal{K}^{\eta,\xi_X}(\delta_j^X) = \mathcal{K}^{\eta,\xi'}(\delta_j^X) = \mathcal{K}^{\eta,\xi'}(\nu X_j.\psi_j^{X \cup \{X_j\}})$. For every $X_p \in X$ it holds by definition of ψ_p^X that $\psi_p^X \equiv \psi_p^{X \cup \{X_j\}}[\nu X_j.\psi_j^{X \cup \{X_j\}}/X_j]$, hence it follows that $\xi'(X_p) = \xi_X(X_p) \stackrel{\text{I.H.}}{=} \mathcal{K}^{\eta,\xi_X}(\nu X_p.\psi_p^X) \stackrel{\text{L.2.15}}{=} \mathcal{K}^{\eta,\xi_X[J/X_j]}(\nu X_p.\psi_p^{X \cup \{X_j\}}) = \mathcal{K}^{\eta,\xi'}(\nu X_p.\psi_p^{X \cup \{X_j\}})$, i.e. ξ' qualifies as a valuation $\xi_{X \cup \{X_j\}}$. Now, using the definition of $\xi'(X_j)$ and $\mathcal{K}_i^{\eta,\xi_X} \models \delta_j^X$, it follows that $\mathcal{K}_i^{\eta,\xi'} \models \psi_j^{X \cup \{X_j\}}$ from which one concludes similarly as above that $\mathcal{K}_i^{\eta,\xi'} \models \delta_l^{X \cup \{X_j\}}$, proving the assertion $(*)$.

Induction Step 2: $i > 0$ and there is some $n_j \in N$ such that $\eta(V(n_j)) \in m_{i-1}(n_j)$ and $(n_j, n_l) \in S$. The proof is analogous using the conjunct of ψ_j^X corresponding to the state transition $(n_j, n_l) \in S$.

Conversely, assume that $\mathcal{K}$ is a temporal structure which is accepted by T. Let again $M = (m_0, m_1, \ldots)$ denote the marking of T over $\mathcal{K}$. It has to be shown that $\mathcal{K} \models (\forall *)\varphi^T$. We show for every set $X \subseteq \{X_0, \ldots, X_k\}$ that

$$(*) \quad \mathcal{K}_i^{\eta,\xi_X} \models \delta_j^X$$

for every $i \geq 0$, every $0 \leq j \leq k$, every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ satisfying $\eta(V(n_j)) \in m_i(n_j)$ and every propositional valuation ξ_X satisfying $\xi_X(X_l) = \{i \mid \eta(V(n_l)) \in m_i(n_l)\}$ for every $X_l \in X$. In particular, hence, $\mathcal{K}_0^{\eta,\xi_\emptyset} \models \delta_j^\emptyset \equiv \varphi^T$ for every valuation η of the rigid subject variables and every propositional valuation $\xi_\emptyset$ (observe that both the conditions on η and $\xi_\emptyset$ are trivial), i.e. $\mathcal{K} \models (\forall *)\varphi^T$. The assertion $(*)$ is proven by induction on $\{X_0, \ldots, X_k\} \setminus X$.

Case 1: $X_j \in X$. Then $\delta_j^X \equiv X_j$ and by the assumption that $\eta(V(n_j)) \in m_i(n_j)$ and the definition of ξ_X, it follows that $i \in \xi_X(X_j)$, i.e. $\mathcal{K}^{\eta,\xi_X} \models \delta_j^X$.

Case 2: $X_j \notin X$. In this case,

$$\begin{aligned} \delta_j^X &\equiv \nu X_j.\psi_j^{X \cup \{X_j\}} \\ &\equiv \nu X_j.\vartheta(n_j) \wedge \bigwedge_{(n_j,n_l)\in I} (\zeta(n_j,n_l) \to \delta_l^{X \cup \{X_j\}}) \wedge \bigwedge_{(n_j,n_l)\in S} \circ\delta_l^{X \cup \{X_j\}} \end{aligned}$$

Let X' denote $X \cup \{X_j\}$. We prove that for every $p \geq 0$ such that $\eta(V(n_j)) \in m_p(n_j)$,

$$(**) \quad \mathcal{K}_p^{\eta,\xi_{X'}} \models \psi_j^{X'},$$

yielding

$$\xi_{X'}(X_j) = \{p \geq 0 \mid \eta(V(n_j)) \in m_p(n_j)\} \subseteq \mathcal{K}^{\eta,\xi_{X'}}(\psi_j^{X'}),$$

hence also

$$\xi_{X'}(X_j) \subseteq \mathcal{K}^{\eta,\xi_{X'}}(\nu X_j.\psi_j^{X'}).$$

Since $\eta(V(n_j)) \in m_i(n_j)$ by the assumption, it follows that $i \in \xi_{X'}(X_j)$, implying $\mathcal{K}_i^{\eta,\xi_{X'}} \models \nu X_j.\psi_j^{X'}$. But the truth of $\nu X_j.\psi_j^{X'}$ in some state does not depend on the valuation of the variable X_j, hence it follows that also $\mathcal{K}_i^{\eta,\xi_X} \models \nu X_j.\psi_j^{X'} \equiv \delta_j^X$, i.e. the assertion $(*)$.

For the proof of $(**)$, assume $p \geq 0$ such that $\eta(V(n_j)) \in m_p(n_j)$.

1. Since T accepts $\mathcal{K}$, it follows that $\mathcal{K}_p^\eta \models \vartheta(n_j)$.

2. Let $(n_j, n_l) \in I$ be an internal edge of T and assume $\mathcal{K}_p^\eta \models \zeta(n_j, n_l)$. By the definition of m_p, it follows that $\eta(V(n_l)) \in m_p(n_l)$, hence by the induction hypothesis for $(*)$ it follows that $\mathcal{K}_p^{\eta,\xi_{X'}} \models \delta_l^{X'}$. Thus, $\mathcal{K}_p^{\eta,\xi_{X'}} \models \zeta(n_j, n_l) \rightarrow \delta_l^{X'}$ holds for every internal edge $(n_j, n_l) \in I$ starting in n_j.

3. Let $(n_j, n_l) \in S$ be a state transition of T. Then $\eta(V(n_l)) \in m_{p+1}(n_l)$, hence by the induction hypothesis for $(*)$ it follows that $\mathcal{K}_{p+1}^{\eta,\xi_{X'}} \models \delta_l^{X'}$, i.e. $\mathcal{K}_p^{\eta,\xi_{X'}} \models \circ\delta_l^{X'}$.

The assertion $(**)$ now follows from the conjunction of 1, 2 and 3 using the definition of $\psi_j^{X'}$. ♠

As a consequence of the preceding lemmas 5.12 and 5.15, transition schemes and (universal closures of) extended program clauses agree in the class of properties they can define. As in the propositional case, it suffices to consider only transition schemes satisfying any combination of the restrictions regularity, normality, completeness or monotonicity defined above. The relevant constructions are straightforward adaptations of the corresponding ones in the propositional case, hence only proof sketches and graphical representations will be given.

Lemma 5.16 *For every transition scheme T there is a regular transition scheme T' such that for every temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$, every formula P, every valuation $\eta : \mathcal{Y} \rightarrow |\mathcal{S}|$ and every $i \geq 0$,*

$$\begin{array}{rl} & \exists n \in N : P \in \vartheta(n), \eta(V(n)) \in m_i(n) \\ \Longleftrightarrow & \exists n' \in N' : P \in \vartheta'(n'), \eta(V'(n')) \in m_i'(n'), \end{array}$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m_0', m_1', \ldots)$ denote the markings of T and T' over $\mathcal{K}$, respectively. In particular, T and T' accept the same temporal structures. Moreover, if T is normal, complete or monotonic, then T' will have the same properties.

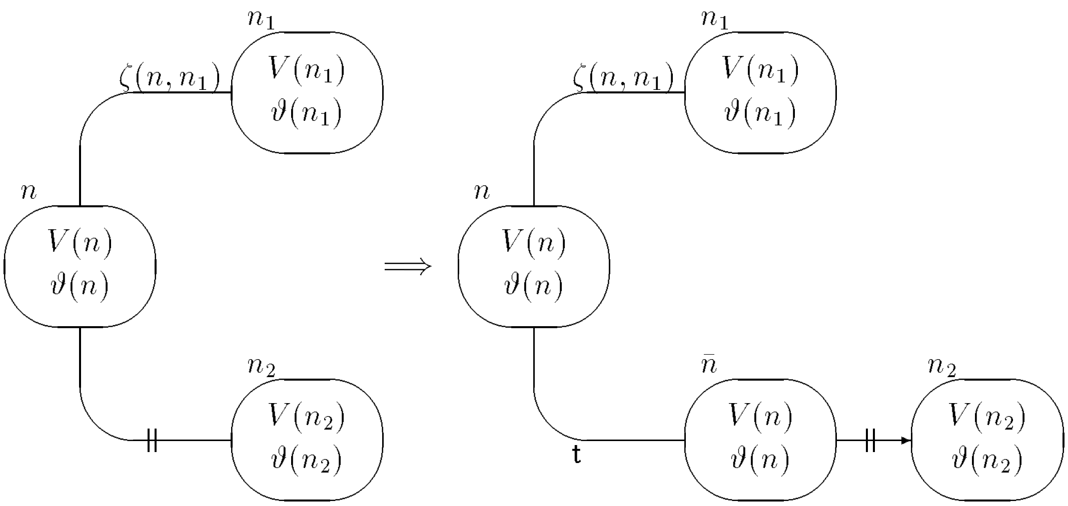


Figure 5.3: Eliminating an irregular node (lemma 5.16)

Proof: See figure 5.3 for the relevant construction step which has to be applied to every node n violating the regularity condition. ♠

Lemma 5.17 *For every transition scheme T there is a normal transition scheme T' such that for every temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$, every formula P, every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ and every $i \geq 0$,*

$$\begin{array}{ll} & \exists n \in N : P \in \vartheta(n), \eta(V(n)) \in m_i(n) \\ \Longleftrightarrow & \exists n' \in N' : P \in \vartheta'(n), \eta(V'(n')) \in m'_i(n'), \end{array}$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over $\mathcal{K}$, respectively. In particular, T and T' accept the same temporal structures. Moreover, if T is regular or monotonic, then T' will have the same properties.

Proof: Construct the conjunctive normal form $D_1 \wedge \ldots \wedge D_k$ (where the D_i are disjunctions of literals) of every (quantifier-free!) edge label $\zeta(n, n')$. This transformation neither changes the set of rigid subject variables appearing in the edge label nor, by the definition of $Bind(A)$ for antecedents A, the set $Bind(\zeta(n, n'))$ of rigid variables the edge label can bind. Hence, $\zeta(n, n')$ can be replaced by its conjunctive normal form without violating the variable conditions for transition schemes. In particular, by $V(n') = V(n) \cup Bind(\zeta(n, n')) = V(n) \cup \bigcap_{i=1}^{k} Bind(D_i)$ it follows that $V(n') \subseteq V(n) \cup Bind(D_i)$ for every $1 \leq i \leq k$, while on the other hand, $Bind(D_i) \subseteq \bigcup_{i=1}^{k} FRV(D_i) \subseteq$

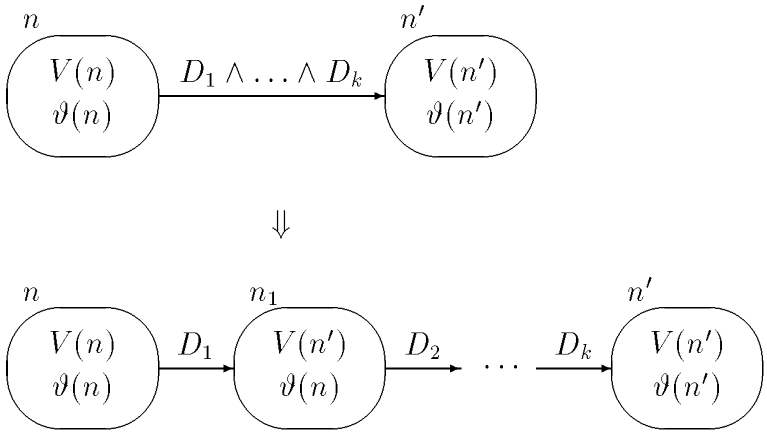


Figure 5.4: Normalizing transition schemes (lemma 5.17)

$V(n) \cup \mathit{Bind}(\zeta(n, n')) = V(n')$ implies that $V(n') \supseteq V(n) \cup \mathit{Bind}(D_i)$ for every $1 \leq i \leq k$. Together it follows that $V(n) \cup \mathit{Bind}(D_i) = V(n) \cup \mathit{Bind}(D_j) = V(n')$ for every $1 \leq i, j \leq k$. Now, replacing every edge whose label includes $k \geq 2$ conjuncts by k edges as indicated in figure 5.4 (where the relevant definitions for the intermediate nodes $n_1, \ldots, n_{k-1}$ are $V'(n_i) = V(n')$, $\vartheta'(n_i) = \vartheta(n)$ and $\zeta'(n_i, n_{i+1}) = D_{i+1}$) results in a transition scheme satisfying all variable conditions which is obviously normal whereas the transformation preserves regularity and monotonicity. Completeness may be obtained by applying the construction of lemma 5.18. ♠

Lemma 5.18 *For every transition scheme T there is a complete transition scheme T' such that for every temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$, every formula P, every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ and every $i \geq 0$,*

$$\begin{array}{rl} & \exists n \in N : P \in \vartheta(n), \eta(V(n)) \in m_i(n) \\ \iff & \exists n' \in N' : P \in \vartheta'(n'), \eta(V'(n')) \in m'_i(n'), \end{array}$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over $\mathcal{K}$, respectively. In particular, T and T' accept the same temporal structures. Moreover, if T is regular, normal or monotonic, then T' will have the same properties.

Proof: For every node n in T, for which $I(n) \neq \emptyset$ and the disjunction $\bigvee_{(n,n^*) \in I} \zeta(n, n^*)$ is not (first-order) valid, add a new node n' where $V'(n') = V(n)$ which is labelled by $\vartheta'(n') = \vartheta(n)$, and an internal edge $I'(n, n')$ inscribed by $\zeta'(n, n') = \mathsf{t}$. Define $I'(n') = S'(n') = \emptyset$. Obviously, the resulting transition graph is complete and satisfies the assertion of the lemma. Moreover, the construction of T' preserves regularity, normality and monotonicity. ♠

Let T be a complete transition scheme. Then for every temporal structure $\mathcal{K}$, every $i \geq 0$, every node $n \in N$ and every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ satisfying $\eta(V(n)) \in m_i^{\mathcal{K}}(n)$ (where $M^{\mathcal{K}} = (m_0^{\mathcal{K}}, m_1^{\mathcal{K}}, \ldots)$ denotes the marking of T over $\mathcal{K}$) there is an I^*-path $\pi = (n^0, n^1, \ldots, n^k)$ such that $n^0 = n$, $(n^j, n^{j+1}) \in I$, $I(n^k) = \emptyset$, $\mathcal{K}_i^\eta \models \zeta(n^j, n^{j+1})$ for $0 \leq j < k$ and $\eta(V(n^j)) \in m_i^{\mathcal{K}}(n^j)$ for $0 \leq j \leq k$. Such a path π will be called a *fullpath for n and η in $m_i^{\mathcal{K}}$*.

Lemma 5.19 *For every transition scheme T there is a monotonic transition scheme T' such that for every temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$, every formula P, every valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ and every $i \geq 0$,*

$$\begin{array}{rl} & \exists n \in N : P \in \vartheta(n), \eta(V(n)) \in m_i(n) \\ \iff & \exists n' \in N' : P \in \vartheta'(n'), \eta(V'(n')) \in m'_i(n'), \end{array}$$

where $M = (m_0, m_1, \ldots)$ and $M' = (m'_0, m'_1, \ldots)$ denote the markings of T and T' over $\mathcal{K}$, respectively. In particular, T and T' accept the same temporal structures. Moreover, if T is regular, normal or complete, then T' will have the same properties.

Proof: The construction for transition schemes is a straightforward adaptation of the corresponding one for propositional transition graphs (cf. lemma 4.19). The relevant construction step for a "maximal" violator of non-monotonicity (as defined in the proof of lemma 4.19) is represented in figure 5.5. Notice that by $V(n_1) \subseteq V(n_2)$, $FRV(\vartheta(n_1)) \subseteq V(n_1)$ and $FRV(\vartheta(n_2)) \subseteq V(n_2)$ which have to hold for the original transition scheme, the condition $FRV(\vartheta(n_1) \cup \vartheta(n_2)) \subseteq V(n_2)$ holds in the node $\bar{n}_2$ of the transformed transition scheme. ♠

As for the corresponding constructions in the propositional case, it should be noted that the transformations described in the lemmas 5.16 – 5.18 preserve the condition of lemma 5.13 regarding the dependency order of state components appearing in node and edge labels of transition schemes associated with proper program clauses. The construction of lemma 5.19 preserves the weaker condition that for every I^+-path $\pi = (n^0, n^1, \ldots, n^k)$ and every module state component κ such that $\kappa \in \mathit{write}(P)$ for some $P \in \vartheta(n^k) \setminus \vartheta(n^{k-1})$, $\lambda < \kappa$ holds for every state component $\lambda \in \mathcal{C}$ which appears in some edge label $\zeta(n^0, n^1), \ldots, \zeta(n^{k-1}, n^k)$ and every $\lambda \in FFV(Q)$ for every $Q \in \vartheta(n^j)$ for $0 \leq j \leq k$.

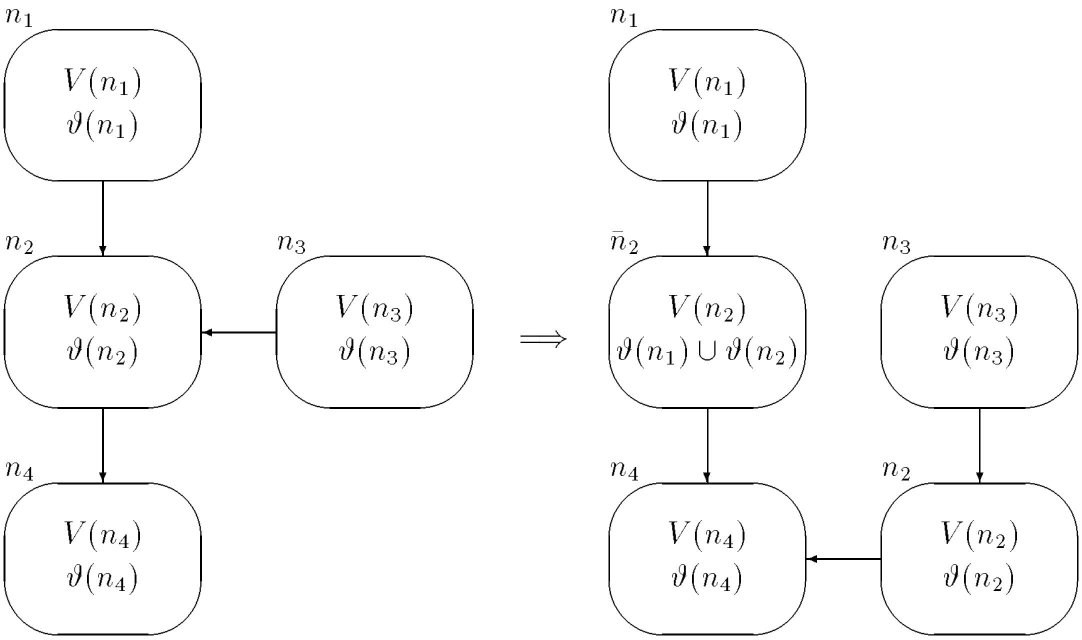


Figure 5.5: Construction step for lemma 5.19

5.3 Conflict schemes

Program clauses will be "executed" by attempting to build a model for them. The technical vehicle for model construction will be the transition schemes introduced in the previous section which, via their node labels, define the extensions of the flexible predicates of $\mathcal{P}_\Pi$ in every state as well as the values assigned to flexible variables $z \in \mathcal{Z}_\Pi$. The syntactic restrictions on (proper) program clauses ensure that finite sets suffice for all predicate extensions in every state (assuming that this condition holds for every environment predicate $P \in \mathcal{P}_E$). Moreover, since only positive literals are allowed in pre-clauses (hence, in node labels), no conflict can arise between formulas $P(t_1, \ldots, t_m)$ and $\neg P(t'_1, \ldots, t'_m)$. However, there still is a chance of conflict for formulas $z = t$ and $z = t'$ assigning values to the same flexible variable: It has to be ensured that both terms evaluate to the same value. This will again be ensured by a condition of compatibility of program clauses which will be prepared by the definition of conflict schemes in the present section.

Definition 5.20 (conflict scheme) *Let $T^1 = (N^1, V^1, I^1, S^1, \vartheta^1, \zeta^1, n_0^1)$ and $T^2 = (N^2, V^2, I^2, S^2, \vartheta^2, \zeta^2, n_0^2)$ be two regular transition schemes where the sets of rigid subject variables appearing in node or edge labels of T^1 and T^2, respectively, are disjoint. The* conflict scheme *$T = (N, V, I, S, \vartheta, \zeta, n_0)$ of T^1 and T^2 is the transition scheme defined as follows:*

- *$N \subseteq N^1 \times N^2$ is the smallest set of nodes satisfying the following conditions:*

- *$n_0 = (n_0^1, n_0^2) \in N$.*

- *If $(n^1, n^2) \in N$ is a node of T and $I^1(n^1) \neq \emptyset$, then $I(n^1, n^2) = I^1(n^1) \times \{n^2\}$ (in particular, $I^1(n^1) \times \{n^2\} \subseteq N$), $S(n^1, n^2) = \emptyset$, and $\zeta((n^1, n^2), (n, n^2)) = \zeta^1(n^1, n)$ for every $(n, n^2) \in I(n^1, n^2)$.*

- *If $(n^1, n^2) \in N$ is a node of T and $I^1(n^1) = \emptyset$ but $I^2(n^2) \neq \emptyset$, then $I(n^1, n^2) = \{n^1\} \times I^2(n^2)$ (in particular, $\{n^1\} \times I^2(n^2) \subseteq N$), $S(n^1, n^2) = \emptyset$, and $\zeta((n^1, n^2), (n^1, n)) = \zeta^2(n^2, n)$ for every $(n^1, n) \in I(n^1, n^2)$.*

- *If $(n^1, n^2) \in N$ is a node of T and $I^1(n^1) = I^2(n^2) = \emptyset$, then $I(n^1, n^2) = \emptyset$ and $S(n^1, n^2) = S^1(n^1) \times S^2(n^2)$ (in particular, $S^1(n^1) \times S^2(n^2) \subseteq N$).*

- *For every node $(n^1, n^2) \in N$, $V(n^1, n^2) = V^1(n^1) \cup V^2(n^2)$.*

- *For every node $(n^1, n^2) \in N$, its label is defined by $\vartheta(n^1, n^2) = \{t = t' \mid z = t \in \vartheta^1(n^1)$ and $z = t' \in \vartheta^2(n^2)$ for some $z \in \mathcal{Z}\}$. The* enriched node label *of (n^1, n^2) is defined as*

$$\vartheta^+(n^1, n^2) = \vartheta(n^1, n^2) \cup \vartheta^1(n^1) \cup \vartheta^2(n^2).$$

Observe that the conflict scheme T of T^1 and T^2 satisfies the variable conditions of definition 5.4:

- For the initial node (n_0^1, n_0^2) of T, $V(n_0^1, n_0^2) = V^1(n_0^1) \cup V^2(n_0^2) = \emptyset$.

- For every formula $t = t' \in \vartheta(n^1, n^2)$ contained in the node label of some node (n^1, n^2), $FRV(t = t') = FRV(t) \cup FRV(t')$ and by the corresponding conditions for T^1 and T^2, it follows that $FRV(t) \subseteq V^1(n^1)$ and $FRV(t') \subseteq V^2(n^2)$, hence $FRV(t = t') \subseteq V^1(n^1) \cup V^2(n^2) = V(n^1, n^2)$. Similar reasoning establishes the condition $FRV(\zeta((n', n''), (n^1, n^2))) \subseteq V(n', n'') \cup Bind(\zeta((n', n''), (n^1, n^2)))$ concerning the rigid subject variables appearing in edge labels.

- Every state transition $((n^1, n^2), (n', n'')) \in S$ results from state transitions $(n^1, n') \in S^1$ and $(n^2, n'') \in S^2$, hence $V^1(n') = V^1(n^1)$ and $V^2(n'') = V^2(n^2)$, implying $V(n', n'') = V^1(n') \cup V^2(n'') = V^1(n^1) \cup V^2(n^2) = V(n^1, n^2)$. The condition $V(n', n'') = V(n^1, n^2) \cup Bind(\zeta((n', n''), (n^1, n^2)))$ for every internal edge $((n', n''), (n^1, n^2)) \in I$ is proven analogously.

When using the tuple interpretation of $V(n)$, we will assume that for every node $(n^1, n^2) \in N$, $V(n^1, n^2)$ is defined as the tuple obtained by concatenating $V^1(n^1)$ and $V^2(n^2)$. This will be denoted as $V(n^1, n^2) = (V^1(n^1), V^2(n^2))$. ♠

The following lemmas establish that acceptance by all relevant conflict schemes is again a necessary and sufficient condition for the model construction to succeed. The chain of reasoning is analogous to the propositional case. The first lemma states that the marking of the conflict scheme over any temporal structure is contained in the "product" of the markings of the original transition schemes.

Lemma 5.21 *Let T^1 and T^2 be two regular transition schemes, and let T be the conflict scheme of T^1 and T^2. Let $\mathcal{K}$ be a temporal structure and denote by M, M^1 and M^2 the markings of T, T^1 and T^2, respectively, over $\mathcal{K}$. Then for every $i \in I\!N$ and all nodes $n^1 \in N^1$, $n^2 \in N^2$ (where $V^1(n^1) = (y_1, \ldots, y_k)$ and $V^2(n^2) = (y'_1, \ldots, y'_l)$) it holds that*

$$m_i(n^1, n^2) \subseteq \{(d_1, \ldots, d_k, d'_1, \ldots, d'_l) \mid \begin{array}{l}(d_1, \ldots, d_k) \in m^1_i(n^1) \text{ and} \\ (d'_1, \ldots, d'_l) \in m^2_i(n^2)\}.\end{array}$$

Proof: by induction on the definition of M: The assertion will be abbreviated by $m_i(n^1, n^2) \subseteq (m^1_i(n^1), m^2_i(n^2))$.

- By definition of M^1 and M^2 it holds that both $m^1_0(n^1_0) = \{()\}$ and $m^2_0(n^2_0) = \{()\}$, and by definition of M, $m_0(n^1_0, n^2_0) = \{()\}$, yielding the assertion.

- Assume the assertion holds for $(n', n'') \in N$ and that $((n', n''), (n^1, n^2)) \in I$ is an internal edge of T and there is some valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ of the rigid subject variables satisfying $\eta(V(n', n'')) \in m_i(n', n'')$ such that $\mathcal{K}^\eta_i \models \zeta((n', n''), (n^1, n^2))$. By the induction hypothesis it follows that $m_i(n', n'') \subseteq (m^1_i(n'), m^2_i(n''))$. By $V(n', n'') = (V(n'), V(n''))$ this implies $\eta(V^1(n')) \in m^1_i(n')$ and $\eta(V^2(n'')) \in m^2_i(n'')$. By the definition of I, either $n' = n^1$ and $\zeta((n', n''), (n^1, n^2)) = \zeta^2(n'', n^2)$ or $n'' = n^2$ and $\zeta((n', n''), (n^1, n^2)) = \zeta^1(n', n^1)$. In either case, $\eta(V^1(n^1)) \in m^1_i(n^1)$ and $\eta(V^2(n^2)) \in m^2_i(n^2)$.

- Assume that $i > 0$ and the assertion holds for some node $(n', n'') \in N$ where $((n', n''), (n^1, n^2)) \in S$ and η is a valuation such that $\eta(V(n', n'')) \in m_{i-1}(n', n'')$. Then by the induction hypothesis, $\eta(V^1(n')) \in m^1_{i-1}(n')$ and $\eta(V^2(n'')) \in m^2_{i-1}(n'')$. By the definition of S it follows that $(n', n^1) \in S^1$ and $(n'', n^2) \in S^2$, implying $\eta(V^1(n^1)) = \eta(V^1(n')) \in m^1_i(n^1)$ and $\eta(V^2(n^2)) = \eta(V^2(n'')) \in m^2_i(n^2)$.

For every $(d_1, \dots, d_k, d'_1, \dots, d'_l) \in m_i(n^1, n^2)$, one of the cases distinguished above applies by the definition of M, proving $m_i(n^1, n^2) \subseteq (m_i^1(n^1), m_i^2(n^2))$. ♠

From this lemma it follows that a temporal structure will be accepted by the conflict scheme of two transition schemes T^1 and T^2 if it is accepted by both T^1 and T^2:

Lemma 5.22 *Let T^1, T^2 and T be defined as before. Every structure which is accepted by both T^1 and T^2 is also accepted by T.*

Proof: Let $\mathcal{K}$ be any temporal structure which is accepted by T^1 and T^2 and let M^1, M^2 and M, as before, denote the markings of T^1, T^2 and T, respectively, over $\mathcal{K}$. Assume that $(n^1, n^2) \in N$ is a node containing the formula $t = t' \in \vartheta(n^1, n^2)$ in its label, and that $\eta : \mathcal{Y} \to \mathcal{S}$ is a valuation of the rigid subject variables satisfying $\eta(V(n^1, n^2)) \in m_i(n^1, n^2)$. We have to show that $\mathcal{K}_i^\eta \models t = t'$. By the construction of T, there is some flexible variable z such that $z = t \in \vartheta^1(n^1)$ and $z = t' \in \vartheta^2(n^2)$. Lemma 5.21 implies (using the definition of $V(n^1, n^2)$) that $\eta(V^1(n^1)) \in m_i^1(n^1)$ and $\eta(V^2(n^2)) \in m_i^2(n^2)$. By the acceptance of $\mathcal{K}$ by T^1 and T^2 it follows that $\mathcal{K}_i^\eta \models z = t$ and $\mathcal{K}_i^\eta \models z = t'$, hence $\mathcal{K}_i^\eta \models t = t'$. ♠

Lemma 5.22 establishes acceptance by the conflict scheme as a necessary condition for the acceptance of some temporal structure by both original transition schemes. Let us now turn to the proof that the conflict scheme of two regular, complete and monotonic transition schemes captures all possible inconsistencies between node labels of the two original transition schemes:

Lemma 5.23 *Let T^1 and T^2 be regular and complete transition schemes, and let T be the conflict scheme of T^1 and T^2. Let $\mathcal{K}$ be a temporal structure and denote by M^1, M^2 and M the markings of T^1, T^2 and T over $\mathcal{K}$, respectively. Let $n' \in N^1$ and $n'' \in N^2$ be two nodes and $\eta : \mathcal{Y} \to \mathcal{S}$ be a valuation satisfying, for some $i \geq 0$, $\eta(V^1(n')) \in m_i^1(n')$ and $\eta(V^2(n'')) \in m_i^2(n'')$, and let $\pi' = (n'_0, \dots, n'_k)$ be a fullpath for n' and η in m_i^1 and $\pi'' = (n''_0, \dots, n''_l)$ be a fullpath for n'' and η in m_i^2. Then there are $0 \leq u \leq k$ and $0 \leq v \leq l$ such that $(n'_u, n''_v) \in N$ and $\eta(V(n'_u, n''_v)) \in m_i(n'_u, n''_v)$.*

Proof: by simultaneous induction on the definition of M^1 and M^2:

- If $i = 0$, and $n' = n_0^1$ and $n'' = n_0^2$ are the initial nodes of T^1 and T^2, respectively, then $(n', n'') = (n_0^1, n_0^2)$ is the initial node of T, and by $V^1(n') = V^2(n'') = V(n', n'') = \emptyset$, the condition on the valuation η is trivially satisfied for $n'_u = n'$ and $n''_v = n''$.

- Assume the assertion holds for nodes $n^1 \in N^1$ and $n'' \in N^2$ where $(n^1, n') \in I^1$ and $\mathcal{K}_i^\eta \models \zeta^1(n^1, n')$ for some valuation η satisfying $\eta(V^1(n^1)) \in m_i^1(n^1)$. Consider the fullpath $\pi^1 = n^1 \circ \pi'$ for n^1 and η in m_i^1 which passes through n'. By the induction hypothesis, for every fullpath π'' for n'' and η in m_i^2, there is some node n''_v appearing in π'' such that either the assertion is satisfied for some $(n'_u, n''_v) \in N$ where n'_u appears in the path π' (in which case we are done), or that $\eta(V(n^1, n''_v)) \in m_i(n^1, n''_v)$. But then, by $I^1(n^1) \neq \emptyset$ it follows from the construction of T that $((n^1, n''_v), (n', n''_v)) \in I$ and $\zeta((n^1, n''_v), (n', n''_v)) = \zeta^1(n^1, n')$, hence $(n', n''_v) \in N$ and $\eta(V(n', n''_v)) \in m_i(n', n''_v)$.

- Assume the assertion holds for nodes $n^2 \in N^2$ and $n' \in N^1$ where $(n^2, n'') \in I^2$ and $\mathcal{K}_i^\eta \models \zeta^2(n^2, n'')$. By repeated applications of the previous argument it follows that either $(n'_k, n''_v) \in N$ and $\eta(V(n'_k, n''_v)) \in m_i(n'_k, n''_v)$ for some $0 \leq v \leq l$ (in which case the assertion also holds for n''), or that $(n'_k, n^2) \in N$ and $\eta(V(n'_k, n^2)) \in m_i(n'_k, n^2)$. Because T^1 is complete and π' is a fullpath for n' and η in m_i^1, it follows that $I^1(n'_k) = \emptyset$, hence by the construction of T, $((n'_k, n^2), (n'_k, n'')) \in I$ from which one concludes similarly as above that $(n'_k, n'') \in m_i$.

- Assume the assertion holds for two nodes $n^1 \in N^1$, $n^2 \in N^2$, and assume that $(n^1, n') \in S^1$ and $(n^2, n'') \in S^2$. Let η be any valuation satisfying $\eta(V^1(n^1)) \in m_i^1(n^1)$ and $\eta(V^2(n^2)) \in m_i^2(n^2)$. Since both T^1 and T^2 are regular, it follows that $I^1(n^1) = \emptyset$ and $I^2(n^2) = \emptyset$, hence both fullpaths for n^1 and n^2 are trivial, and the induction hypothesis yields $(n^1, n^2) \in N$ and $\eta(V(n^1, n^2)) \in m_i(n^1, n^2)$. By the construction of T it follows that $((n^1, n^2), (n', n'')) \in S$, hence $(n', n'') \in N$ and by $V(n', n'') = V(n^1, n^2)$ and the definition of m_{i+1}, moreover $\eta(V(n', n'')) \in m_{i+1}(n', n'')$. ♠

Let us now assume that T^1 and T^2 are also monotonic and that there are two "assignments" $z = t$ and $z = t'$ of terms t, t' to the same flexible variable z occurring in node labels of two nodes $n^1 \in N^1$ and $n^2 \in N^2$. Then lemma 5.23 implies that for every valuation η "admissible" for both nodes in some state of the markings of the original transition schemes, the terms t and t' denote the same value if the conflict scheme accepts the temporal structure:

Lemma 5.24 *Let T^1 and T^2 be two normal, regular, complete and monotonic transition schemes, and let T be the conflict scheme of T^1 and T^2. Let $\mathcal{K}$ be a temporal structure accepted by T and denote by M^1, M^2 and M the markings of T^1, T^2 and T over $\mathcal{K}$, respectively. Assume $n^1 \in N^1$ and $n^2 \in N^2$ are two nodes such that, for some flexible variable z and terms t, t', $z = t \in \vartheta^1(n^1)$ and $z = t' \in \vartheta^2(n^2)$, and let $\eta_1, \eta_2 : \mathcal{Y} \to \mathcal{S}$ be valuations satisfying $\eta_1(V^1(n^1)) \in m_i^1(n^1)$ and $\eta_2(V^2(n^2)) \in m_i^2(n^2)$. Then $\mathcal{K}_i^{\eta_1}(t) = \mathcal{K}_i^{\eta_2}(t')$.*

Proof: By the definition of T, all rigid subject variables appearing in node labels of T^1 and T^2, respectively, are pairwise different. Hence, the new valuation η defined by $\eta(y) = \eta_1(y)$ for every variable y appearing in the node label $\vartheta^1(n)$ of some node $n \in N^1$, and $\eta(y) = \eta_2(y)$ else is well-defined. Let $\pi^1 = (n_0^1, \ldots, n_k^1)$ and $\pi^2 = (n_0^2, \ldots, n_l^2)$ be two fullpaths for n^1 and η in m_i^1, and n^2 and η in m_i^2, respectively. By lemma 5.23, there are $0 \leq u \leq k$ and $0 \leq v \leq l$ such that $(n_u^1, n_v^2) \in N$ and $\eta(V(n_u^1, n_v^2)) \in m_i(n_u^1, n_v^2)$. Since T^1 and T^2 were assumed to be monotonic transition schemes, it follows that $z = t \in \vartheta^1(n_u^1)$ and $z = t' \in \vartheta^2(n_v^2)$, hence by the definition of T, $t = t' \in \vartheta(n_u^1, n_v^2)$. By the acceptance of $\mathcal{K}$ by T it follows that $\mathcal{K}_i^{\eta_1}(t) = \mathcal{K}_i^{\eta}(t) = \mathcal{K}_i^{\eta}(t') = \mathcal{K}_i^{\eta_2}(t')$. ♠

5.4 Normal models and program completion

It has been indicated at the beginning of the present chapter that for the construction of models of first-order νTL programs an approach will be taken which resembles the closed-world view of logic programming and database theory (cf., e.g., Gallaire, Minker & Nicolas [19]) in that only those object tuples are contained in the extension of a predicate (at some state) whose inclusion can explicitly be proven. The purpose of this section is to give a formal definition of such "normal" interpretations as well as to characterize them by additional νTL axioms which can be schematically generated for every first-order νTL program. These axioms re-establish the correspondence between the operational semantics of a νTL program under this special execution strategy and the declarative semantics of the (augmented) program which may be important for proving properties of the program as it is executed by the interpreter.

For the definition of normal models of νTL programs, the representation of program clauses by transition schemes again turns out to be very helpful:

Definition 5.25 (normal models) *Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a first-order νTL program (i.e. a finite set of proper program clauses) and let $T^1, \ldots, T^q$ denote the associated transition schemes for $\varphi^1, \ldots, \varphi^q$ according to algorithm 5.10. A temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$ is called a* normal model *of Φ if*

- *$\mathcal{K}$ is a model of the universal closures $(\forall *)\varphi^1, \ldots, (\forall *)\varphi^q$ of the program clauses of Φ and*
- *$\mathcal{K}$ satisfies the* normality condition *for every (m-place) predicate symbol $P \in \mathcal{P}_\Pi$ representing a module state component: For every $i \geq 0$ and all $(d_1, \ldots, d_m) \in |\mathcal{S}|^m$, $(d_1, \ldots, d_m) \in \sigma_i(P)$ only if for some $1 \leq j \leq$*

q there exists a node $n \in N^j$, a formula $P(t_1, \ldots, t_m) \in \vartheta^j(n)$ and a valuation $\eta : \mathcal{Y} \to \mathcal{S}$ of the rigid subject variables such that $\eta(V(n)) \in m_i^j(n)$ and $\mathcal{K}_i^\eta(t_p) = d_p$ for $1 \le p \le m$ (where $M^j = (m_0^j, m_1^j, \ldots)$ denotes the marking of the transition scheme T^j over $\mathcal{K}$).

Normal models can be axiomatically characterized by simulating the definition of the marking of the transition schemes using νTL formulas. These formulas require an extension of the first-order language $\mathcal{L}$ of the given program by additional predicate symbols used to represent the sets $m_i(n)$ of object tuples bound in the i-th state of the marking of a transition scheme T at node n.

Definition 5.26 (transition axioms) *Let $T = (N, V, I, S, \vartheta, \zeta, n_0)$ be a transition scheme whose set of nodes is $N = \{n_0, \ldots, n_k\}$ where n_0 is the initial node of T. For every node n_l, introduce a new predicate symbol M_l whose arity is given by the number $|V(n_l)|$ of rigid subject variables bound at node n_l. (In particular, the predicate symbol M_0 corresponding to the initial state n_0 of T is a proposition letter.) For $V(n_l) = (y_1, \ldots, y_k)$, denote the formula $M_l(y_1, \ldots, y_k)$ by $M_l(V(n_l))$.*

Denote by $F \subseteq N$ the set of nodes of the transition scheme T which can be reached by an I^*-path from the initial node n_0 of T. Then the transition axioms for *T are defined as*

$(C0)$ M_0

$(C1)$ $(\forall *)\, (M_l(V(n_l)) \leftrightarrow \bigvee_{n_k \in F, (n_k, n_l) \in I} (M_k(V(n_k)) \wedge \zeta(n_k, n_l)))$ *for* $n_0 \neq n_l \in F$

$(C2)$ $(\forall *)\, \neg M_l(V(n_l))$ *for* $n_l \notin F$

$$(C3) \quad \Box(\forall *)\, (\circ M_l(V(n_l)) \leftrightarrow \bigvee_{(n_k, n_l) \in I} \circ(M_k(V(n_k)) \wedge \zeta(n_k, n_l)) \vee \bigvee_{(n_k, n_l) \in S} M_k(V(n_k)) \quad \textit{for every } n_l \in N$$

(where empty disjunctions are, as always, identified with f*).*

Example 5.27 The following formulas constitute the transition axioms for the transition scheme of figure 5.1:

$$
\begin{array}{ll}
(C0) & M_0 \\
(C1) & \forall x\,(M_1(x) \leftrightarrow M_0 \wedge P(x)) \\
 & \forall x, y\,(M_2(x,y) \leftrightarrow M_1(x) \wedge Q(x,y)) \\
 & \forall x, y\,(M_3(x,y) \leftrightarrow M_2(x,y) \wedge \neg S(y)) \\
 & \forall x, y\,(M_4(x,y) \leftrightarrow M_2(x,y) \wedge S(y)) \\
(C2) & \text{(no instances)} \\
(C3) & \Box(\circ M_0 \leftrightarrow \mathsf{f}) \\
 & \Box\forall x\,(\circ M_1(x) \leftrightarrow \circ(M_0 \wedge P(x))) \\
 & \Box\forall x, y\,(\circ M_2(x,y) \leftrightarrow \circ(M_1(x) \wedge Q(x,y)) \vee M_4(x,y)) \\
 & \Box\forall x, y\,(\circ M_3(x,y) \leftrightarrow \circ(M_2(x,y) \wedge \neg S(y))) \\
 & \Box\forall x, y\,(\circ M_4(x,y) \leftrightarrow \circ(M_2(x,y) \wedge S(y)))
\end{array}
$$

♠

Lemma 5.28 *Let T be a transition scheme and let $\mathcal{K} = (\mathcal{S}, \sigma)$ be a model of the transition axioms for T. Denote by $M = (m_0, m_1, \ldots)$ the marking of T over $\mathcal{K}$. Then for every $i \geq 0$ and every node $n_l \in N$,*

$$\sigma[i](M_l) = m_i(n_l)$$

Proof: by induction on i and the length of the longest I^*-path ending in node n_l:

Assume $V(n_l) = (y_1, \ldots, y_m)$ and $(d_1, \ldots, d_m) \in \sigma[i](M_l)$. If $i = 0$, then by $(C2)$ it follows that $n_l \in F$. If $n_l = n_0$ is the initial node of T then by definition of m_0 and by $(C0)$, $m_0(n_0) = \{()\} = \sigma[0](M_0)$. Else, by $(C1)$, for every valuation η satisfying $\eta(y_1) = d_1, \ldots, \eta(y_m) = d_m$, there is some $n_k \in F$ such that $(n_k, n_l) \in I$ and $\mathcal{K}_0^\eta \models \zeta(n_k, n_l)$. Let, w.l.o.g., $V(n_k) = (y_1, \ldots, y_p)$ for some $p \leq m$ (since $V(n) \subseteq V(n')$ is implied by the variable conditions for transition schemes). Then by the induction hypothesis it follows that $(d_1, \ldots, d_p) \in m_0(n_k)$, implying $(d_1, \ldots, d_m) \in m_0(n_l)$ by the definition of the marking M and by $\mathcal{K}_0^\eta \models \zeta(n_k, n_l)$. For $i > 0$, the proof is similar using axiom $(C3)$ instead of $(C1)$. Hence, $\sigma[i](M_l) \subseteq m_i(n_l)$.

Conversely, assume $(d_1, \ldots, d_m) \in m_i(n_l)$. If $i = 0$ and $n_l = n_0$ is the initial node of T, then $\mathcal{K}_0 \models M_0$ by axiom $(C0)$, i.e. $\sigma[0](M_0) = \{()\} = m_0(n_0)$. If there is some $n_k \in N$ such that $(n_k, n_l) \in I$ and $\mathcal{K}_i^\eta \models \zeta(n_k, n_l)$ for some valuation η satisfying $\eta(V(n_k)) \in m_i(n_k)$ then by induction hypothesis it follows that $\eta(V(n_k)) \in \sigma[i](M_k)$, i.e. $\mathcal{K}_i^\eta \models M_k(V(n_k))$, implying $\eta(V(n_l)) \in \sigma[i](M_l)$ by $(C1)$ or $(C3)$. If $i > 0$ and there is some $n_k \in N$ such that $(n_k, n_l) \in S$ and some valuation η satisfying $\eta(V(n_k)) \in m_{i-1}(n_k)$, the proof is similar using the second disjunct of axiom $(C3)$. It follows that $m_i(n_l) \subseteq \sigma[i](M_l)$, completing the proof of the lemma. ♠

Using the transition axioms, normal models of νTL programs can now be characterized axiomatically. To this end, the completion of a νTL program is defined similarly as for classical logic programs (cf. LLOYD [35]).

Definition 5.29 (program completion) *Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a νTL program and denote by $T^1, \ldots, T^q$ the associated transition schemes for $\varphi^1, \ldots, \varphi^q$ according to algorithm 5.10 where the set of nodes of T^j be $N^j = \{n_0^j, \ldots, n_{k_j}^j\}$. For every m-place predicate symbol $P \in \mathcal{C}_\Pi$ modelling a module state component and $1 \leq j \leq q$, define the set*

$$Def_j(P) = \{n \in N^j \mid P(t_1, \ldots, t_m) \in \vartheta^j(n) \text{ for some terms } t_1, \ldots, t_m\}$$

of nodes of T^j containing some formula $P(t_1, \ldots, t_m)$ in their node label and the degree $\deg(P)$ *of P by*

$$\deg(P) = \max\{1 \leq j \leq q \mid Def_j(P) \neq \emptyset\}.$$

Let $y_1, \ldots, y_m \in \mathcal{Y}$ be "new" rigid subject variables which do not occur in any node or edge labels of any transition scheme T^j. Now, the completion axiom *(DEF_P) is defined as*

$$(DEF_P) \quad \Box\forall y_1, \ldots, y_m \, (P(y_1, \ldots, y_m) \rightarrow \bigvee_{j=1}^{\deg(P)} \bigvee_{n_l^j \in Def_j(P)} \exists V^j(n_l^j) \, (M_l^j(V^j(n_l^j)) \wedge \bigvee_{P(t_1,\ldots,t_m) \in \vartheta^j(n_l^j)} (y_1 = t_1 \wedge \ldots \wedge y_m = t_m))).$$

*A state component $\lambda \in \mathcal{C}_\Pi$ is called k-*defined *if for no state component $\mu \leq \lambda$ (where $\leq$ denotes the reflexive closure of the dependency order $<$) and no node $n \in N^j$ for $k+1 \leq j \leq q$ there is a formula $F \in \vartheta^j(n)$ such that $\mu \in write(F)$.*

*For $1 \leq k \leq q$, the k-*completion of Φ *is defined as the set of formulas containing the universal closures $(\forall *)\varphi^1, \ldots, (\forall *)\varphi^k$ of the program clauses in Φ, the transition axioms for $T^1, \ldots, T^k$ and the formulas (DEF_P) for every k-defined predicate symbol P. The* completion of Φ *is defined as the q-completion of Φ.*

Examples of completion axioms will be given in the discussion of application examples in chapter 7.

Lemma 5.30 *Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a νTL program and let $\mathcal{K} = (\mathcal{S}, \sigma)$ be a model of the k-completion of Φ (for some $1 \leq k \leq q$). Then $\mathcal{K}$ is a model of $\{(\forall *)\varphi^1, \ldots, (\forall *)\varphi^k\}$ which satisfies the normality condition for every k-defined predicate symbol $P \in \mathcal{P}_\Pi$. In particular, models of the completion of Φ are normal models of Φ.*

Proof: If $\mathcal{K}$ is a model of the k-completion of Φ, then it is by definition a model of the universal closures of the program clauses $\varphi^1, \ldots, \varphi^k$ of Φ.

Let $T^1, \ldots, T^q$ denote the transition schemes associated with the program clauses $\varphi^1, \ldots, \varphi^q$ and let $M^j = (m_0^j, m_1^j, \ldots)$ denote, as always, the marking of T^j over $\mathcal{K}$. Assume that, for some $i \geq 0$ and $(d_1, \ldots, d_m) \in |\mathcal{S}|^m$, $(d_1, \ldots, d_m) \in \sigma[i](P)$. By axiom (DEF$_P$) and because P is k-defined, it follows that there is some $1 \leq j \leq k$, a node n_l^j, a formula $P(t_1, \ldots, t_m) \in \vartheta^j(n_l^j)$ and a valuation η of the rigid subject variables such that $\mathcal{K}_i^\eta \models M_l^j(V(n_l^j))$ and $\mathcal{K}_i^\eta(t_p) = d_p$ for $1 \leq p \leq m$. By lemma 5.28 it follows that $\eta(V(n_l^j)) \in m_i^j(n_l^j)$. ♠

Similarly as in the propositional case, the condition of compatibility can now be formulated for first-order νTL programs. Its definition is based on the acceptance of models of subsets $\{\varphi^1, \ldots, \varphi^k\}$ of the program by the conflict schemes of (transition schemes associated with) φ^j and φ^{k+1} for $1 \leq j \leq k+1$ (where for $j = k+1$ two copies of the transition scheme T^{k+1} using disjoint sets of rigid variables have to be employed). Again, the set of models to check for acceptance by the conflict schemes can be restricted to extensions of a set $\mathcal{U}$ of environment structures representing certain environment assumptions. Since at the end we will only be interested in normal models, sometimes the set of models to check can be restricted further using the completion axiom (DEF$_P$) (which is the counterpart to the condition of normality by the above lemma). Observe, however, that the normality condition is "non-compositional": It can only be defined for the program as a whole by considering all transition schemes making up the program. Hence, when appealing to the completion axiom (DEF$_P$) for some predicate symbol P at the stage of testing the $(k+1)$-st program clause φ^{k+1} in order to exclude some non-normal models of $\{\varphi^1, \ldots, \varphi^k\}$ which might not be accepted by the conflict scheme of some T^j and T^{k+1}, P has to be k-defined — a restriction on the program clauses $\varphi^{k+1}, \ldots, \varphi^q$. This requirement seems to be contrary to the stepwise refinement framework sketched in section 4.1 since at the stage of entering some clause φ^{k+1}, the remaining program clauses will in general not be known. On the other hand, it does not appear unreasonable to assume the development process to proceed hierarchically, i.e. to start with the definition of the behavior of some basic module state components on which the components defined later rely. And this hierarchy view is in line with the concept of k-definedness: If it is necessary to know that $P(d_1, \ldots, d_m)$ does not hold to allow a model to be constructed for the program $\{\varphi^1, \ldots, \varphi^{k+1}\}$ and every environment structure satisfying the environment assumptions, then the clause to be added depends on the definition of P, hence P should be k-defined. This is not to say, of course, that the requirement of k-definedness might not be a technical inconvenience to the developer at some times.

In the sequel we will always assume a fixed first-order interpretation $\mathcal{S}$ for the models to be constructed, hence a temporal structure $\mathcal{K} = (\mathcal{S}, \sigma)$ will be identified with the state sequence $\sigma = (\sigma_0, \sigma_1, \ldots)$. Let $\mathcal{U}$ be a set of *environment structures*, i.e. every $\mathcal{E} \in \mathcal{U}$ is an ω-sequence of valuations $(\varepsilon_0, \varepsilon_1, \ldots)$ assigning values $\varepsilon_i(z)$ to flexible variables $z \in \mathcal{Z}_E$ and sets $\varepsilon_i(P) \subseteq |\mathcal{S}|^m$ to every m-place predicate symbol $P \in \mathcal{P}_E$ modelling state components of the environment. A temporal structure $\mathcal{K}$ is called an *extension* of $\mathcal{E} \in \mathcal{U}$ (denoted by $\mathcal{K} \sqsupseteq \mathcal{E}$) if $\sigma_i(\kappa) = \varepsilon_i(\kappa)$ for every $i \geq 0$ and every environment state component $\kappa \in \mathcal{C}_E$.

Definition 5.31 (compatibility) *Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a first-order νTL program and $\mathcal{U}$ a set of environment structures. $T^1, \ldots, T^q$ denote regular, complete and monotonic transition schemes associated with $\varphi^1, \ldots, \varphi^q$, respectively. Φ is called $\mathcal{U}$-*compatible *if for every $0 \leq k < q$ and every environment structure $\mathcal{E} \in \mathcal{U}$, every model $\mathcal{K} \sqsupseteq \mathcal{E}$ of the k-completion of Φ is accepted by the conflict schemes of T^{k+1} and T^j (for every $1 \leq j \leq k+1$).*

Again, $\mathcal{U}$ will usually be defined by a νTL formula Ψ stating the environment assumptions. The compatibility condition can then be stated as follows: For every $0 \leq k < q$, the νTL formulas associated with the conflict schemes of T^{k+1} and T^j (for every $j = 1, \ldots, k+1$) follow from Ψ, the universal closures $(\forall *)\varphi^1, \ldots, (\forall *)\varphi^k$ of the first k program clauses, the transition axioms for $T^1, \ldots, T^k$ and the formulas (DEF_P) for every k-defined predicate symbol $P \in \mathcal{P}_\Pi$. Both versions of the compatibility condition are equivalent due to the lemmas 5.12, 5.15 and 5.30.

5.5 Interpreting νTL programs

The condition of $\mathcal{U}$-compatibility will be shown to guarantee, for every environment structure $\mathcal{E} \in \mathcal{U}$, the existence of a normal model for the νTL-program Φ as an extension of $\mathcal{E}$. For the following interpreter algorithm to be able to actually generate (prefixes of) such a model it is necessary that every predicate modelling an environment component has a finite extension in every state of the environment structure. For the effective evaluation of terms appearing in node or edge labels of transition schemes, the underlying first-order structure $\mathcal{S}$ is assumed to be represented by appropriate data types and algorithms outside of the νTL program. In practice, this requirement can be achieved most easily and flexibly by implementing the following model construction algorithm in a programming language executed by an interpreter which allows appropriate and problem-specific function definitions modelling the intended first-order structure.

The model construction algorithm is a rather straightforward generalization of the corresponding one for propositional νTL programs. The normality conditions will be easy to implement by treating extensions of predicate symbols as database relations and represent only positive information.

Algorithm 5.32 (interpreter for first-order νTL programs)
Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a first-order νTL program and $\mathcal{E} = (\varepsilon_0, \varepsilon_1, \ldots)$ be an environment structure assigning a finite extension $\varepsilon_i(P) \subseteq |\mathcal{S}^m|$ to every (m-place) predicate symbol $P \in \mathcal{P}_E$ modelling an environment component. Denote by $T^k = (N^k, V^k, I^k, S^k, \vartheta^k, \zeta^k, n_0^k)$, for $1 \leq k \leq q$, the normal transition scheme associated with the program clause φ^k according to algorithm 5.10 and lemma 5.17. The construction of the temporal structure $\mathcal{K}(\Phi, \mathcal{E})$, identified with its state sequence $(\sigma_0, \sigma_1, \ldots)$, is described in the following. During the construction of the current state σ_i, the pseudo-value $\sigma_i(z) = \bot$ denotes that the value $\sigma_i(z)$ assigned to the flexible variable z is still undefined. Interpretations $\sigma_i(z)$ and $\sigma_i(P)$ of flexible variables and predicate symbols will at first be stored as (tuples of) terms together with a *substitution* for the rigid variables appearing in these terms, and will be evaluated at some later stage of the construction. The need for such a scheme of delayed evaluation will be motivated in the proof of theorem 5.33. A substitution ρ is given as a finite set (or list) of pairs $d_1/y_1, \ldots, d_p/y_p$ where $d_1, \ldots, d_p \in |\mathcal{S}|$ are (object) values and $y_1, \ldots, y_p \in \mathcal{Y}$ are rigid variables (ρ will sometimes be abbreviated as $\vec{d}/\vec{y}$). The evaluation of a term t under a substitution $\rho = \{d_1/y_1, \ldots, d_p/y_p\}$ in some (partially defined) state σ_i, denoted as $[\![t]\!]_\rho$, is inductively defined as follows:

$$\begin{array}{rcl}
[\![y_j]\!]_\rho & = & d_j \quad \text{for } 1 \leq j \leq p, \\
[\![z]\!]_\rho & = & \left\{ \begin{array}{ll} d & \text{if } \sigma_i(z) = d \in |\mathcal{S}| \\ [\![t]\!]_{\rho'} & \text{if } \sigma_i(z) = (t, \rho') \end{array} \right\} \text{if } z \in \mathcal{Z} \text{ and } \sigma_i(z) \neq \bot, \\
[\![f(t_1, \ldots, t_m)]\!]_\rho & = & \mathcal{S}(f)([\![t_1]\!]_\rho, \ldots, [\![t_m]\!]_\rho) \quad \text{if all } [\![t_i]\!]_\rho \text{ are defined.}
\end{array}$$

A *configuration* of $T^1, \ldots, T^q$ consists of a q-tuple $(\nu^1, \ldots, \nu^q)$ of *bindings* where every ν^k assigns a finite set $\nu^k(n_l^k) \subseteq |\mathcal{S}|^{|V^k(n_l^k)|}$ of object tuples to every node $n_l^k \in N^k$, and a set Exp representing the edges of $T^1, \ldots, T^q$ which still have to be inspected. For two nodes n, n' connected by an internal edge $(n, n') \in I^k$, the expression $L \in \zeta^k(n, n')$ denotes the condition that the literal L appears as a disjunct in the edge label $\zeta^k(n, n')$ of the (normal) transition scheme T^k. The algorithm defines a sequence of configurations $(\kappa_i^0, \kappa_i^1, \ldots)$ for every state σ_i of the temporal structure $\mathcal{K}(\Phi, \mathcal{E})$.

Initialization: Set $i := 0$ and $j := 0$ and define $(\nu^1, \ldots, \nu^q)$ by

$$\begin{array}{rcl}
\nu^k(n_0^k) & = & \{()\}, \\
\nu^k(n_l^k) & = & \emptyset \quad \text{for } n_0^k \neq n_l^k \in N^k
\end{array}$$

for every $1 \leq k \leq q$.

State construction: Assume that $(\nu^1, \dots, \nu^q)$ is the initial q-tuple of bindings for the state σ_i and let

$$Exp := \bigcup_{k=1}^{q} \{(n, n', L) \mid \nu^k(n) \neq \emptyset, (n, n') \in I^k, L \in \zeta^k(n, n')\}$$

be the initial set of edges to inspect. Define, for every flexible variable $z \in \mathcal{Z}$, the (initial) valuation $\sigma_i(z)$ as follows:

$$\sigma_i(z) := \left\{ \begin{array}{cl} \varepsilon_i(z) & \text{if } z \in \mathcal{Z}_E \text{ is an environment component,} \\ (t, \vec{d}/V^k(n)) & \text{if } z = t \in \vartheta^k(n) \text{ for some } n \in N^k \text{ (where} \\ & 1 \leq k \leq q),\ \nu^k(n) \neq \emptyset \text{ and } \vec{d} \in \nu^k(n) \text{ is an} \\ & \text{arbitrary tuple,} \\ \perp & \text{else} \end{array} \right\}$$

and, for every (m-place) predicate symbol $P \in \mathcal{P}$,

$$\sigma_i(P) := \left\{ \begin{array}{cl} \varepsilon_i(P) & \text{if } P \in \mathcal{P}_E \\ \bigcup_{k=1}^{q} \bigcup_{n \in N^k} & \{((t_1, \vec{d}/V^k(n)), \dots, (t_m, \vec{d}/V^k(n))) \mid \\ & \vec{d} \in \nu^k(n), P(t_1, \dots, t_m) \in \vartheta^k(n)\} \quad \text{else} \end{array} \right\}.$$

Observe that at this point, except for environment state components, only (tuples of) terms together with substitutions for rigid variables are stored (once again, the tuple interpretation of $V^k(n)$ has been used). Their evaluation is delayed to ensure that all flexible variables appearing in these terms have actually been defined.

Complete the construction of σ_i as follows:

1. While $Exp \neq \emptyset$, iterate the following steps:

   (a) Set $j := j + 1$. Choose some $(n_k, n'_k, L) \in Exp$ (where $n_k, n'_k \in N^k$) such that L is minimal among all literals appearing in Exp in the sense that for no $(n_l, n'_l, L') \in Exp$ it holds that $\lambda < \mu$ for every state component $\lambda \in \mathit{read}(L')$ and some state component $\mu \in \mathit{read}(L)$.[2] For every flexible variable $z \in \mathcal{Z}_\Pi$ such that $z \leq \kappa$ for some state component $\kappa \in \mathit{read}(L)$ (where $\leq$ denotes the reflexive closure of the dependency order) and $\sigma_i(z) = \perp$, define $\sigma_i(z) := d \in |\mathcal{S}|$ as an arbitrary value.

[2]Observe that such a triple $(n_k, n'_k, L) \in Exp$ has to exist because Exp is a finite set whose size is bounded by $\sum_{k=1}^{q} \sum_{(n,n') \in I^k} |\zeta^k(n, n')|$ where $|\zeta^k(n, n')|$ denotes the number of disjuncts of the edge label $\zeta^k(n, n')$ of the normal transition scheme T^k.

If $L \equiv P(t_1, \ldots, t_m)$ or $L \equiv \neg P(t_1, \ldots, t_m)$ and $\sigma_i(P)$ is still stored in symbolic form, i.e. as a set of term/substitution tuples, force the evaluation of $\sigma_i(P)$, i.e. update $\sigma_i(P)$ by

$$\sigma_i(P) := \{([\![t'_1]\!]_\rho, \ldots, [\![t'_m]\!]_\rho) \mid ((t'_1, \rho), \ldots, (t'_m, \rho)) \in \sigma_i(P)\}.$$

It will be observed in the proof of theorem 5.33 that all term evaluations $[\![t]\!]_\rho$ necessary for the definition of $\sigma_i(P)$ yield defined values.

(b) Assume that $V^k(n_k) = (y_1, \ldots, y_p)$ and $V^k(n'_k) = (y_1, \ldots, y_r)$ $=: \vec{y}$ where $r \geq p$. (Observe that a real implementation of the interpreter algorithm has to employ some sort of indexing to match corresponding rigid subject variables in $V^k(n_k)$ and $V^k(n'_k)$. This will be ignored here to simplify the presentation.) Let $(\nu^1, \ldots, \nu^q)$ denote the bindings of the configuration κ_i^{j-1}, and define the bindings of κ_i^j as $(\nu^1, \ldots, \bar{\nu}^k, \ldots, \nu^q)$ where $\bar{\nu}^k(n) := \nu^k(n)$ for every $n \neq n'_k$, and $\bar{\nu}^k(n'_k) := \nu^k(n'_k) \cup D$ where the set D is defined as follows, depending on the syntactic form of L (more constructive definitions of D will be given in the proof of theorem 5.33):

If $L \equiv P(t_1, \ldots, t_m)$ then

$$\begin{aligned} D := \{\vec{d} = (d_1, \ldots, d_r) \mid ([\![t_1]\!]_{\vec{d}/\vec{y}}, \ldots, [\![t_m]\!]_{\vec{d}/\vec{y}}) \in \sigma_i(P) \\ \text{and } (d_1, \ldots, d_p) \in \nu^k(n_k)\}. \end{aligned}$$

If $L \equiv t_1 = t_2$ then

$$\begin{aligned} D := \{\vec{d} = (d_1, \ldots, d_r) \mid [\![t_1]\!]_{\vec{d}/\vec{y}} = [\![t_2]\!]_{\vec{d}/\vec{y}} \\ \text{and } (d_1, \ldots, d_p) \in \nu^k(n_k)\}. \end{aligned}$$

If $L \equiv \neg P(t_1, \ldots, t_m)$ then $Bind(L) = \emptyset$ and by the variable conditions for the transition scheme T^k it follows that $V^k(n_k) = V^k(n'_k) = (y_1, \ldots, y_p)$. Then

$$D := \{\vec{d} = (d_1, \ldots, d_p) \in \nu^k(n_k) \mid ([\![t_1]\!]_{\vec{d}/\vec{y}}, \ldots, [\![t_m]\!]_{\vec{d}/\vec{y}}) \notin \sigma_i(P)\}.$$

Similarly, if $L \equiv t_1 \neq t_2$ then $V^k(n_k) = V^k(n'_k) = (y_1, \ldots, y_p)$ and

$$D := \{\vec{d} = (d_1, \ldots, d_p) \in \nu^k(n_k) \mid [\![t_1]\!]_{\vec{d}/\vec{y}} \neq [\![t_2]\!]_{\vec{d}/\vec{y}}\}.$$

In either case, if $D \neq \emptyset$ then update σ_i by defining

$$\sigma_i(z) = (t, \vec{d}/V^k(n'_k)) \quad \text{if } z = t \in \vartheta^k(n'_k) \text{ for an arbitrary tuple } \vec{d} \in \bar{\nu}^k(n'_k) \neq \emptyset$$

(this step is necessary only if $\sigma_i(z) = \perp$ has held before), and

$$\sigma_i(P) = \sigma_i(P) \cup \bigcup_{\vec{d} \in D} \{((t_1, \vec{d}/V^k(n'_k)), \ldots, (t_m, \vec{d}/V^k(n'_k))) \mid P(t_1, \ldots, t_m) \in \vartheta^k(n'_k)\}$$

for every predicate symbol P, and update the set Exp of edges still to be inspected to

$$\begin{array}{rcl} Exp & := & (Exp \setminus \{(n_k, n'_k, L)\}) \cup \\ & & \{(n'_k, n''_k, L'') \mid (n'_k, n''_k) \in I^k, L'' \in \zeta^k(n'_k, n''_k)\}. \end{array}$$

2. For every flexible variable $z \in \mathcal{Z}$ such that $\sigma_i(z) = \perp$, define $\sigma_i(z) := d \in |\mathcal{S}|$ as an arbitrary value. Force the evaluation of all flexible variables and predicate extensions in σ_i, i.e. define

$$\begin{array}{rcl} \sigma_i(z) & := & [\![t]\!]_\rho \quad \text{if } \sigma_i(z) = (t, \rho), \quad \text{and} \\ \sigma_i(P) & := & \{([\![t_1]\!]_\rho, \ldots, [\![t_m]\!]_\rho) \mid ((t_1, \rho), \ldots, (t_m, \rho)) \in \sigma_i(P)\}. \end{array}$$

3. Assume that the bindings in the final configuration for the construction of σ_i (i.e. the first configuration κ_i^j where $Exp = \emptyset$) are $(\nu^1, \ldots, \nu^q)$. Then define the initial bindings for the construction of σ_{i+1} to be $(\bar{\nu}^1, \ldots, \bar{\nu}^q)$ where

$$\bar{\nu}^k(n) := \bigcup_{(n', n) \in S^k} \nu^k(n')$$

for $1 \leq k \leq q$. Set $i := i + 1$ and $j := 0$ and repeat the state construction starting with these bindings.

Again, the algorithm works incrementally and is backtracking-free. The following theorem shows that if Φ is $\mathcal{U}$-compatible, the temporal structure $\mathcal{K}(\Phi, \mathcal{E})$ is well-defined for every environment structure $\mathcal{E} \in \mathcal{U}$ and is indeed a normal model of Φ where every state of $\mathcal{K}(\Phi, \mathcal{E})$ is computed in a finite number of steps and is finitely representable:

Theorem 5.33 *Let $\Phi = \{\varphi^1, \ldots, \varphi^q\}$ be a $\mathcal{U}$-compatible first-order νTL program for some set $\mathcal{U}$ of environment structures, and let $\mathcal{E} = (\varepsilon_0, \varepsilon_1, \ldots) \in \mathcal{U}$. Then $\mathcal{K}(\Phi, \mathcal{E})$ is a well-defined temporal structure which is a normal model of Φ. If, moreover, for every $i \geq 0$, the extension $\varepsilon_i(P)$ of every predicate symbol $P \in \mathcal{P}_E$ modelling an environment state component is finite, then the same condition holds for every predicate symbol P and the construction of every state σ_i terminates after a finite number of steps.*

Proof: Let us first prove that $\mathcal{K}(\Phi, \mathcal{E})$ is a normal model of Φ under the assumption that it is well-defined. A straightforward induction on i and j shows that under this assumption the following assertion $(*)$ holds for every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$, every $k = 1, \ldots, q$ and every node $n \in N^k$ (ignoring, again, the indexing of variables):

$$(*) \quad \nu^k(n) \subseteq m_i^k(n) \subseteq \nu^k(n) \cup \{\nu^k(n') \times |\mathcal{S}|^{|V^k(n) \setminus V^k(n')|} \mid (n', n) \in (I^k)^+, \exists n'', L : (n', n'', L) \in Exp\},$$

where $M^k = (m_0^k, m_1^k, \ldots)$ denotes the marking of T^k over $\mathcal{K}(\Phi, \mathcal{E})$. For the proof of $(*)$ it is important to note that for every state component $\lambda \in \mathit{read}(L)$, $\sigma_i(\lambda)$ does not change anymore where L denotes the literal of the triple (n_k, n'_k, L) selected in step (1a) at the configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$. This is obvious for environment components $\lambda \in \mathcal{C}_E$. For a state component $\lambda \in \mathcal{C}_\Pi$, $\sigma_i(\lambda)$ can only change at some later stage of the construction algorithm (where the current configuration is, say, $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^q), Exp'$ for $m > j$ if either $\lambda \equiv z$ is a flexible variable and $\sigma_i(z)$ is chosen arbitrarily at stage (1a) or (2) — which cannot happen for $m > j$ by the assumption that $\lambda \in \mathit{read}(L)$, or the set D computed in step (1b) for the then current triple (n_l, n'_l, L') is nonempty, and n'_l contains some formula $z = t$ (if $\lambda \equiv z$ is a flexible variable) or $P(t_1, \ldots, t_m)$ (if $\lambda \equiv P$ is a predicate symbol). Clearly, the set Exp current at the earlier stage contains some triple (n''_l, n'''_l, L'') such that $(n''_l, n'_l) \in (I^l)^+$. But then, lemma 5.13 implies that $\mu < \lambda$ holds for every $\mu \in \mathit{read}(L'')$, contradicting the condition on the choice of the triple (n_k, n'_k, L).

In particular, when $Exp = \emptyset$ it follows from $(*)$ that $\nu^k(n) = m_i^k(n)$ for every $n \in N^k$. Hence, in this case it follows for every valuation η satisfying $\vec{d} := \eta(V(n)) \in m_i^k(n)$ that $\vec{d} \in \nu^k(n)$ and therefore, by construction of $\mathcal{K}(\Phi, \mathcal{E})$ and the variable condition $FRV(F) \subseteq V^k(n)$ for every formula $F \in \vartheta^k(n)$ that $\mathcal{K}_i^\eta \models P(t_1, \ldots, t_m)$ and $\mathcal{K}_i^\eta \models z = t$ for every such formula appearing in the node label $\vartheta^k(n)$. Hence, $\mathcal{K}(\Phi, \mathcal{E})$ is accepted by T^k and is therefore a model of φ^k by lemma 5.12. The normality of $\mathcal{K}(\Phi, \mathcal{E})$ is immediate by the definition of $\sigma_i(P)$ for every predicate symbol P and the invariant $(*)$.

To show that $\mathcal{K}(\Phi, \mathcal{E})$ is well-defined, let us first convince ourselves that all term evaluations $[\![t]\!]_\rho$ occurring during any state construction yield defined values. During the main interpreter loop, i.e. at step 1, all terms which have to be evaluated occur in the literal L of the triple (n_k, n'_k, L) chosen at step (1a) where it has been ensured by the assignment of arbitrary values that $\sigma_i(z) \neq \bot$ for every flexible variable $z \leq \lambda$ for every $\lambda \in \mathit{read}(L)$ (in particular, $\sigma_i(z) \neq \bot$ for every $z \in \mathit{read}(L)$). The variable conditions for transition schemes ensure that every rigid variable occurring in L is among those in $V^k(n'_k) = \{y_1, \ldots, y_r\}$, hence their evaluation is defined. If $\sigma_i(z) = (t, \rho)$ for some flexible variable $z \in \mathit{read}(L)$, by the construction of $\mathcal{K}(\Phi, \mathcal{E})$ there is some node $n^l \in N^l$

(for some $1 \leq l \leq q$) such that $z = t \in \vartheta^l(n^l)$ and $\rho = \vec{d}/V^l(n^l)$ for some $\vec{d} \in \nu^l(n^l)$. Hence, the variable conditions again ensure that, first, all rigid variables occurring in t are among those in $V^l(n^l)$, i.e. their evaluation is defined and second, that $z' < z$ holds for every flexible variable z' occurring in t, hence again $\sigma_i(z') \neq \perp$ by step (1a). Proceeding inductively, it follows that the evaluation of every term occurring in the chosen literal L terminates, producing some defined value. (Of course, it is crucial that no partial functions are assumed to occur as interpretations $\mathcal{S}(f)$ of function symbols f.)

Similar reasoning applies to the evaluation of terms at stage 2 of algorithm 5.32, since $\sigma_i(z) \neq \perp$ is then guaranteed for every flexible variable z. It should be remarked here that an actual implementation of the algorithm would profitably store every value $\sigma_i(z) \in |\mathcal{S}|$ which has been evaluated once for later reuse. Observe, however, that the delayed evaluation specified by algorithm 5.32 is in general necessary since flexible variables appearing in $FFV(F)$ for formulas $F \in \vartheta(n)$ contained in node labels of the transition schemes representing the program Φ do not have to be defined at the stage of computing the binding $\nu(n)$ of the node n in step (1b) of the algorithm. As a simple example, consider the clause $\Box(z_1 = f(z_2) \wedge (q \rightarrow z_2 = 17) \wedge (\neg q \rightarrow z_2 = -3))$ where $\sigma_i(z_2)$, depending on the truth of q in the current state σ_i (q may either be an environment component or be defined by some other clause of the program) is defined only after defining $\sigma_i(z_1) = (f(z_2), ())$. The requirements placed on program clauses w.r.t. the dependency order, as well as the choice of the triple (n_k, n'_k, L) at step (1a), ensure that every flexible variable required for the evaluation of the terms occurring in the literal L either has been defined, or else it will never be defined during the construction of σ_i and may therefore be chosen arbitrarily. This follows from the assertions (a) and (b) below.

By step 2 of the algorithm 5.32 every flexible variable z and every (m-place) predicate symbol P is eventually assigned some value $\sigma_i(z) \in |\mathcal{S}|$ or $\sigma_i(P) \subseteq |\mathcal{S}|^m$, respectively. Moreover, the assignments to $\sigma_i(P)$ at stage (1b) can not introduce inconsistencies because the sets are only enlarged. The only possible inconsistencies could arise from assignments to $\sigma_i(z) \in |\mathcal{S}|$ for some flexible variable z. The following assertions show that this does not happen:

(a) If, for some $i \geq 0$, $\sigma_i(z)$ is chosen arbitrarily in step (1a) then for every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$ and every node $n \in N^l$ (for $1 \leq l \leq q$) containing a formula $z = t \in \vartheta^l(n)$ it holds that $\nu^l(n) = \emptyset$.

(b) For every $i \geq 0$ and every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$: If $\vec{d} \in \nu^k(n)$ for some node $n \in N^k$ containing a formula $z = t \in \vartheta^k(n)$ then for every configuration $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^q), Exp'$, every node $n' \in N^l$ such that some formula $z = t' \in \vartheta^l(n')$ and every $\vec{c} \in \bar{\nu}^l(n')$ it holds that $[\![t]\!]_{\vec{d}/V^k(n)} = [\![t']\!]_{\vec{c}/V^l(n')}$ if $\sigma_i(z') \neq \perp$ for every $z' < z$.

Proof sketches for these assertions are as follows:

(a) Assume that $z \leq \lambda$ holds for some state component $\lambda \in \mathit{read}(L)$ where (n_k, n'_k, L) is the triple chosen at some stage of the algorithm, the current configuration is $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$, the valuation $\sigma_i(z)$ is chosen arbitrarily in step (1a), and that $n \in N^l$ is some node containing a formula $z = t \in \vartheta^l(n)$. It has to be shown that for every configuration $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^q), Exp'$ it holds that $\bar{\nu}^l(n) = \emptyset$. By the construction of $\mathcal{K}(\Phi, \mathcal{E})$ this condition certainly holds for $m < j$. Assume that $\bar{\nu}^l(n) \neq \emptyset$ for some (earliest) configuration κ_i^m where $m \geq j$. This can only happen if the current triple at that stage is of the form (n', n, L') which in turn implies that Exp contains some triple (n'', n''', L'') such that $(n'', n) \in (I^l)^+$. By lemma 5.13 it follows that $\mu < z$, hence also $\mu < \lambda$, for every $\mu \in \mathit{read}(L'')$, contradicting the condition on the choice of the triple (n_k, n'_k, L).

(b) Assume, to the contrary, that for some smallest pair $1 \leq k \leq l \leq q$ there is some smallest $i \geq 0$ such that there exist configurations $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$ and $\kappa_i^m = (\bar{\nu}^1, \ldots, \bar{\nu}^q), Exp'$ (where, w.l.o.g., $m \geq j$) and nodes $n \in N^k$, $n' \in N^l$ such that $z = t \in \vartheta^k(n)$, $z = t' \in \vartheta^l(n')$ and for some $\vec{d} \in \nu^k(n)$, $\vec{c} \in \bar{\nu}^l(n')$ it holds that $\sigma_i(z') \neq \perp$ for every $z' < z$ and $[\![t]\!]_{\vec{d}/V^k(n)} \neq [\![t']\!]_{\vec{c}/V^l(n')}$. By the choice of k and l, the prefix of $\mathcal{K}(\Phi, \mathcal{E})$ computed up to (but excluding) configuration κ_i^m is well-defined and admits an extension by continuing algorithm 5.32 starting in the configuration just before κ_i^m but taking into consideration only the program $\Phi' = \{\varphi^1, \ldots, \varphi^{l-1}\}$, resulting in a model $\mathcal{K}'$ for Φ' satisfying the normality condition for all $(l-1)$-defined predicate symbols. Since Φ is $\mathcal{U}$-compatible, $\mathcal{K}'$ is accepted by the conflict scheme of (regular, complete and monotonic versions of) T^k and T^l. Hence, by lemma 5.24 (and using lemmas 5.16, 5.18 and 5.19), it cannot hold that both $\vec{d} \in m_i^k(n)$ and $\vec{c} \in m_i^l(n')$ which would, however, follow from $\vec{d} \in \nu^k(n)$ and $\vec{c} \in \bar{\nu}^l(n')$ by the invariant $(*)$ above.

Assume now that $\varepsilon_i(P)$ is finite for every $i \geq 0$ and every environment predicate symbol $P \in \mathcal{P}_E$. It remains to show that $\sigma_i(P)$ is a finite set for every $i \geq 0$ and every predicate symbol $P \in \mathcal{P}$ as well as the finiteness of $\nu^k(n)$ for every configuration $\kappa_i^j = (\nu^1, \ldots, \nu^q), Exp$ and every $n \in N^k$. Since $\mathit{write}(F) \subseteq \mathcal{C}_\Pi$ for every formula $F \in \vartheta^k(n)$ this assertion only remains to be shown for predicate symbols $P \in \mathcal{P}_\Pi$ modelling module state components. The proof runs by induction on the algorithm (i.e. by induction on i and j). First, the assertion clearly holds for the bindings of the initial configuration κ_0^0 and, by the induction hypothesis and the definition of the bindings of κ_{i+1}^0 in step 3, also for every configuration κ_{i+1}^0 which establishes the finiteness of $\sigma_i(P)$ at the initialization of every state construction. The crucial step is to show that the set D defined in step (1b) is finite. Let (n_k, n_k', L) be the current triple. For negative literals $L \equiv \neg P(t_1, \ldots, t_m)$ or $L \equiv t_1 \neq t_2$ the finiteness of D follows directly from the induction hypothesis since in this case $D \subseteq \nu^k(n^k)$. Assume $L \equiv P(t_1, \ldots, t_m)$ where $V^k(n_k) = (y_1, \ldots, y_p) =: \vec{y}$ and $V^k(n_k') = (y_1, \ldots, y_r)$ (possibly after suitable renumbering). By the variable condition on T^k it follows that $Bind(L) = \{y_{p+1}, \ldots, y_r\}$, hence these variables appear in L among the terms t_j, say, w.l.o.g., $t_1 \equiv y_{p+1}, \ldots, t_l \equiv y_r$ (where $l = r - p$). The remaining terms $t_{l+1}, \ldots, t_m$ only contain rigid subject variables from among $y_1, \ldots, y_p$. By the induction hypothesis, the current value of $\sigma_i(P) \subseteq |\mathcal{S}|^m$ is finite — observe that the evaluation possibly performed in step (1a) can only reduce the size $|\sigma_i(P)|$, assume $\sigma_i(P) = \{\vec{d}_1, \ldots, \vec{d}_s\}$ where every $\vec{d}_j = (d_j^1, \ldots, d_j^m)$, and, moreover, $\nu^k(n_k)$ is finite, say, $\nu^k(n_k) = \{\vec{c}_1, \ldots, \vec{c}_t\}$ where $\vec{c}_j = (c_j^1, \ldots, c_j^p)$. Now,

$$D = \{(c_j^1, \ldots, c_j^p, d_i^1, \ldots, d_i^l) \mid (d_i^1, \ldots, d_i^l, [\![t_{l+1}]\!]_{\vec{c}_j/\vec{y}}, \ldots, [\![t_m]\!]_{\vec{c}_j/\vec{y}}) \in \sigma_i(P)\}$$

clearly is finite (and its computation requires at most $s * t * (m - l)$ term evaluations and $s * t$ tests for inclusion, hence it is effectively computable). The argument for $L \equiv t_1 = t_2$ is similar.

For the proof of termination of the construction of every single state σ_i, observe as in the propositional case that

$$|\bigcup_{k=1}^{q}\{(I^k)^+(n) \mid \exists(n, n', L) \in Exp\}| + |Exp|$$

decreases with every iteration (and that the computation of every set D requires a finite number of steps). However, unlike the propositional case, the complexity of every single iteration pass cannot be bounded by a constant (such as the number of module-owned state components) but depends essentially on the size of the predicate extensions $\varepsilon_i(P)$ defined by the environment structure $\mathcal{E}$ and the complexity of the terms to be evaluated for the computations of the sets D and the updates to $\sigma_i(P)$ and $\sigma_i(z)$. ♠

For a reasonably efficient implementation of this interpreter algorithm for first-order νTL programs, optimization techniques known from relational database technology should be considered. For example, the definition of the set D resembles a join operation (see DATE [15]) where the "relations" involved (for $L \equiv P(t_1, \ldots, t_m)$) are $\nu^k(n_k)$ and $\sigma_i(P)$.

The reader might wonder why the underlying first-order structure has been confined to providing the universe and the interpretation for function symbols.

Frequently it will seem just as sensible to define (rigid) predicates on the data objects and use them as tests in antecedents (though certainly not to define such predicates via formulas in node labels). The reason for this restriction is that all predicate extensions were required to be finite, and this will generally not be the case for standard predicates on the underlying domains. However, the definitions can be extended in a quite straightforward way to handle such predicates provided they are only used as tests. In particular, they must not be allowed to bind rigid variables. In any case, the restricted definitions are easier to handle, and any such standard predicates can be simulated by functions.

For a complete example for the development of a module for a reactive systems using first-order νTL and resulting in a program which can be interpreted by the above algorithm see chapter 7.

Chapter 6

Combining modules

6.1 A model for distributed systems

In the preceding two chapters, a reactive system was viewed as a monolithic program without further structure. This was reflected in a rather coarse distinction between state components owned (or "written") by the program Φ on one hand and by its environment on the other. However, reactive systems are frequently made up of a number of independent, loosely coupled modules (think, e.g., of distributed data bases or network operating systems), and the behavior of the entire system is produced by a cooperation of those single modules taking place (ideally, at least) over well-defined interfaces. This model of distributed reactive systems has already been used in the discussion of section 4.1 and will be central to the present chapter. A structured approach to system development where the single modules are developed largely independently and are later combined to implement the specification of the entire system should be supported by any proposed development method.

The correctness assertions concerning a single module are expressed by a set of properties of the module. One issue to address when composing modules is how to derive properties of the combined system from properties of single modules. This transfer of properties should be as simple as possible and completely schematic (ideally, the composition preserves those properties which are shared by all modules — in this case a specification method is usually called *compositional*).

Another issue which has to be dealt with in this context concerns the mutual discharge of environment assumptions. It has already been indicated in section 4.1 that the environment of every module consists of (the relevant parts of) the environment of the entire system plus interface components owned by other modules. Recall that during the development of single modules environment assumptions could be introduced, and that the success of the model construction algorithm depended on the environment meeting these assumptions. During the composition of several modules M_j (given as νTL programs Φ_j), it must be shown that the parts of the environment assumption $\mathcal{U}_j$ of a module M_j which refer to components owned by the other modules M_i, $i \neq j$, of the system are met in every run of the combined system. It might seem that this simply amounts to showing that the conjunction $\bigwedge_{i \neq j} \Phi_i$ implies (the relevant part of) $\mathcal{U}_j$ — but then one is caught in a cycle if the modules mutually depend on one another. Nevertheless, ABADI & LAMPORT [3] have stated a rule which suspiciously reminds of cyclic reasoning and have identified conditions guaranteeing its soundness. However, the rule has been stated very abstractly and the conditions of applicability have been given using complicated semantical notions for which no obvious syntactical counterparts exist (moreover, the usage of the term "property" is more restricted than simply referring to sets of state sequences — specifically, closure under stuttering is required). A similar rule will be introduced below and applied to νTL programs.

Syntax of composition

Let us first study the derivation of properties of composed systems in a quite general framework which will be applied to νTL programs in section 6.4. Assume the structure of a reactive system to be described as a set of modules $M_1, \ldots, M_n$ (where $n \geq 2$), identifying the state components owned by each module M_j (cf. the discussion in section 4.1) as well as the state components of the module's environment which M_j may read. It will be assumed in the sequel that every system state component is owned (i.e. "written") by exactly one module. This assumption simplifies the presentation but the main ideas would also apply without it. If several modules have to write to the same component, this should be implemented by introducing a separate module owning the component which accepts requests to change the component's value. For every module M_j, Π_j denotes the set of state components owned by M_j, and E_j, the set of state components owned by the environment of M_j which may be read by the module. In particular, it holds that $E_j \cap \Pi_j = \emptyset$ and that $\Pi_i \cap \Pi_j = \emptyset$ for $1 \leq i \neq j \leq n$. Then the parallel composition $M = M_1 \parallel \ldots \parallel M_n$ is well-formed where the sets E and Π of environment and system state components of M are defined by

$$\Pi = \bigcup_{i=1}^{n} \Pi_i \quad \text{and} \quad E = (\bigcup_{i=1}^{n} E_i) \setminus \Pi.$$

Observe, in particular, that again $E \cap \Pi = \emptyset$. Let $\mathcal{L}_j = E_j \cup \Pi_j$ denote the "language" of module M_j, and Σ_j the state space of module M_j, i.e. the set of valuations of state components $\kappa \in \mathcal{L}_j$. Simultaneously, $\mathcal{L}_j$ can (and will) be viewed as a propositional or first-order logical language in which to formulate νTL formulas expressing properties of M_j. Notice that the languages $\mathcal{L}_i$ and $\mathcal{L}_j$, for $i \neq j$, need not be distinct, but that they may share environment components, and that module components of one module may appear as environment components of the other. By Σ_j^E (resp., Σ_j^Π), we denote the restriction of Σ_j to the environment (resp., module) components of M_j. Similarly, $\mathcal{L}$ and Σ (Σ^E, Σ^Π) denote language and state space (restricted to environment or system components), respectively, of the system. If $s \in \Sigma$, denote by $s{\downarrow_j}$ the restriction of s to state components $\kappa \in \mathcal{L}_j$. Moreover, define the *augmented language* $\hat{\mathcal{L}} = E \cup \Pi \cup \{sch_1, \ldots, sch_n\}$ where sch_j, for $1 \leq j \leq n$, are new proposition letters which are not contained in $E \cup \Pi$, and the *augmented state space* $\hat{\Sigma}$ as the set of valuations of elements of $\hat{\mathcal{L}}$.

An interleaving semantics for composition

Associated with the definition of every module M_j in some formal framework there will be an operational semantics identifying possible runs of M_j which we will assume to be defined — in our general model — via a transition relation $\delta_j \subseteq \Sigma_j^* \times \Sigma_j$. Intuitively, δ_j indicates the possible successor states given a finite computation history. Formally, a *run* of M_j is defined as an infinite state sequence $\rho = (r_0, r_1, \ldots) \in \Sigma_j^\omega$ such that for every $i \geq 0$, $(\rho[..i-1], \rho[i]) \in \delta_j$. (In particular, $(\varepsilon, r_0) \in \delta_j$ has to hold for the initial state r_0 of ρ.) A finite state sequence $\rho = (r_0, \ldots, r_m) \in \Sigma_j^*$ is called a *partial run* of M_j if $(\rho[..i-1], \rho[i]) \in \delta_j$ holds for every $0 \leq i \leq m$.

Runs of the composed system $M = M_1 \parallel \ldots \parallel M_n$ will be $\hat{\Sigma}^\omega$-sequences defined as interleavings of runs of the single modules. Some further notation will be required: Let $\sigma = (s_0, s_1, \ldots) \in \hat{\Sigma}^* \cup \hat{\Sigma}^\omega$ be a state sequence over the augmented state space of M. Define the *M_j-projection of σ* as the sequence $\sigma{\downarrow_j} = (s_{n_0}{\downarrow_j}, s_{n_1}{\downarrow_j}, \ldots) \in \Sigma_j^* \cup \Sigma_j^\omega$ where $(s_{n_0}, s_{n_1}, \ldots)$ is the subsequence of states of σ satisfying $s_{n_i}(sch_j) = \{()\}$ (observe that $\sigma{\downarrow_j}$ can be finite, even empty, for an infinite state sequence $\sigma \in \hat{\Sigma}^\omega$).

Definition 6.1 (runs of composed systems) *Let $M = M_1 \parallel \ldots \parallel M_n$ be a system composed from the modules M_j where, for $1 \leq j \leq n$, a transition relation $\delta_j \subseteq \Sigma_j^* \times \Sigma_j$ has been defined.*

A state sequence $\sigma = (s_0, s_1, \ldots) \in \hat{\Sigma}^* \cup \hat{\Sigma}^\omega$ is a (partial) run *of M if the following conditions hold:*

(i) *For $0 \leq i < |\sigma|$, $s_i(sch_j) = \{()\}$ for at most one $1 \leq j \leq n$,*

(ii) *For* $1 \leq i < |\sigma|$, $s_i(\lambda) = s_{i-1}(\lambda)$ *for every state component* $\lambda \in \Pi_j$ *if* $s_i(sch_j) = \emptyset$, *and*

(iii) *For every* $1 \leq j \leq n$, $\sigma{\downarrow_j}$ *is a (partial) run of* M_j.

A schedule *for* M *is a function* $sch : \mathbb{N} \to \{0, 1, \ldots, n\}$ *(intuitively indicating which module causes the transition to the i-th state where* $sch(i) = 0$ *corresponds to idle transitions).*

For a (partial) run σ *of* M, *the schedule* $sch : \mathbb{N} \to \{0, 1, \ldots, n\}$ *is said to* generate σ *if, for* $0 \leq i < |\sigma|$, $sch(i) = 0$ *if* $\sigma[i](sch_j) = \emptyset$ *for every* $1 \leq j \leq n$ *and* $sch(i) = j$ *if* $\sigma[i](sch_j) = \{()\}$ *otherwise.*

This definition of a run of a system $M = M_1 \parallel \ldots \parallel M_n$ reflects the interleaving semantics of parallel composition where modules take turns in making a step. Notice that no general fairness assumption has been made — even runs where some or even all modules never take a turn are considered admissible. Fairness assumptions will rather be stated for individual modules whenever they are needed. Moreover, an infinite state sequence $\sigma \in \hat{\Sigma}^\omega$ is a run of M iff every finite prefix $\sigma' \prec \sigma$ is a partial run of M, hence the property of being a run of M is a safety property over $\hat{\Sigma}$ in the sense of definition 2.1.

6.2 Deriving properties of composed systems

The first issue to be addressed is how properties of a composed system $M = M_1 \parallel \ldots \parallel M_n$ can be derived from properties which have been established for single modules M_j. This question will be studied for properties expressed as νTL formulas where the language $\mathcal{L}_j$ associated with module M_j will also be assumed to be the logical language used for expressing properties of M_j (and similarly, $\hat{\mathcal{L}}$-formulas express properties of M). As in chapter 5, the underlying first-order structure $\mathcal{S}$ is assumed to be fixed outside of the program. In particular, runs $\sigma \in \Sigma_j^\omega$ of M_j can also be viewed as temporal structures over which to interpret $\mathcal{L}_j$-formulas (and similarly for $\sigma \in \hat{\Sigma}^\omega$).

First, there are some obvious properties of runs of composed systems similar to the (structural) program axioms of Kröger [29] which follow from the conditions of definition 6.1 and may be used in a formal system for proving properties of such systems:

Lemma 6.2 *Let* $M = M_1 \parallel \ldots \parallel M_n$ *be a composed system. Then, for every* $1 \leq i \neq j \leq n$, *every m-place predicate symbol* P *and every flexible variable* z

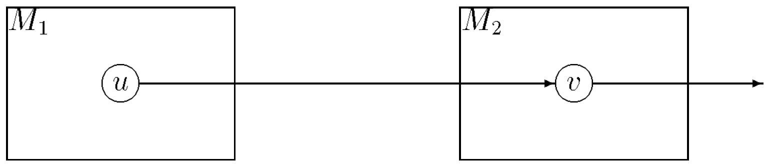


Figure 6.1: A simple distributed reactive system

owned by module M_j (i.e. $P \in \Pi_j$ or $z \in \Pi_j$) and every run $\sigma \in \hat{\Sigma}^\omega$ of M, the following formulas hold:

$$
\begin{array}{ll}
(P0) & \Box(sch_j \to \neg sch_i) \\
(P1) & \Box \forall x_1 \ldots x_m \, (P(x_1, \ldots, x_m) \to \circ(P(x_1, \ldots, x_m) \; \mathsf{unless} \; sch_j)), \\
(P2) & \Box \forall x (z = x \to \circ(z = x \; \mathsf{unless} \; sch_j)).
\end{array}
$$

Proof: The first axiom holds because of the condition that, for every run $\sigma = (s_0, s_1, \ldots)$ and every $i \geq 0$, $s_i(sch_j) = \{()\}$ for at most one $1 \leq j \leq n$.

Let $(\eta(x_1), \ldots, \eta(x_m)) \in s_i(P)$ for some $i \geq 0$ and some valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ of the rigid subject variables, where $\sigma = (s_0, s_1, \ldots)$ is a run of M. If $s_k(sch_j) = \emptyset$ for every $k > i$ it follows that $s_k(P) = s_i(P)$ for every $k > i$ which implies $\sigma_i^\eta \models \circ(P(x_1, \ldots, x_m) \; \mathsf{unless} \; sch_j)$. Else, there is some smallest index $k > i$ such that $s_k(sch_j) = \{()\}$. Then $(\eta(x_1), \ldots, \eta(x_m)) \in \sigma_l(P)$ for every $i < l < k$ as asserted by $(P1)$.

The proof of the third axiom is analogous. ♠

To see what can be expected for the translation of properties of single modules into properties of composed systems, let us first study a simple example:

Example 6.3 Consider the reactive system represented in figure 6.1 which consists of two modules M_1 and M_2 owning the (flexible) variables u and v, respectively. Assume that module M_2 simply copies the value of u into v in such a way that v receives the value read in the previous state, i.e. M_2 has the property $\Box \forall y (u = y \to \circ v = y)$. This property will not hold for the system $M_1 \parallel M_2$. Instead, for every run σ of $M_1 \parallel M_2$ where $\sigma{\downarrow_2} = (s_{n_0}, s_{n_1}, \ldots)$ it will hold that $s_{n_k}(v) = s_{n_{k-1}}(u)$ for $1 \leq k < |\sigma{\downarrow_2}|$. This property can be expressed by the $\hat{\mathcal{L}}$-formula $\Box \forall y (u = y \wedge sch_2 \to \circ(v = y \; \mathsf{atnext} \; sch_2))$.

Another property of M_2 (which is implied by the first one) is expressed by $\Diamond u = 0 \to \Diamond v = 0$. This property fails in $M_1 \parallel M_2$ for two distinct reasons:

- M_2 might not observe the value $u = 0$, i.e. for some run σ of M, $\sigma \models \Box\neg(u = 0 \wedge sch_2)$. This problem can be taken care of by weakening the formula, similarly as above, to $\Diamond(u = 0 \wedge sch_2) \rightarrow \Diamond v = 0$.
- Even if M_2 observes the value $u = 0$, it might not be scheduled after this state, i.e. $\sigma \models \circ\Box\neg sch_2$ atnext $(u = 0 \wedge sch_2)$. This possibility can be ruled out if suitable fairness assumptions are adopted. ♠

The indications how to relativize properties of single modules in order to obtain properties of composed systems gleaned from the above example generalize, yielding the following translation τ_j taking $\mathcal{L}_j$-formulas F in positive normal form to $\hat{\mathcal{L}}$-formulas $\tau_j(F) = \tau_j^*(F)$ atnext sch_j where $\tau_j^*(F)$ is inductively defined by

$$\begin{array}{rcl} \tau_j^*(L) & = & L \quad \text{for every literal } L \\ \tau_j^*(X) & = & X \quad \text{for every proposition variable } X \in \mathcal{X} \\ \tau_j^*(F \; op \; G) & = & \tau_j^*(F) \; op \; \tau_j^*(G) \quad \text{for } op \in \{\wedge, \vee\} \\ \tau_j^*(\circ F) & = & \circ(\tau_j^*(F) \; \mathsf{atnext} \; sch_j) \\ \tau_j^*(Qy\; F) & = & Qy\; \tau_j^*(F) \quad \text{for } Q \in \{\forall, \exists\} \\ \tau_j^*(\sigma X.F) & = & \sigma X.\tau_j^*(F) \quad \text{for } \sigma \in \{\mu, \nu\} \end{array}$$

Example 6.4 For the formula $P \equiv \Box\forall y(u = y \rightarrow \circ v = y) \equiv \nu X.(\forall y(u \neq y \vee \circ v = y)) \wedge \circ X$ which expresses a property of module M_2 of the above example, the translation yields $\tau_2(P) = (\nu X.(\forall y(u \neq y \vee \circ(v = y \; \mathsf{atnext} \; sch_2))) \wedge \circ(X \; \mathsf{atnext} \; sch_2))$ atnext sch_2. By the following lemma, this formula expresses a property of the system $M_1 \parallel M_2$. Additionally, every run of $M_1 \parallel M_2$ satisfies the axioms of lemma 6.2. Applying similar reasoning as in example 6.6 below, $\tau_2(P)$ may be strengthened with the help of these axioms to obtain the formula $\Box\forall y(u = y \wedge sch_2 \rightarrow \circ(v = y \; \mathsf{atnext} \; sch_2))$.

In the case of the second property of the above system which was expressed by $Q \equiv \Diamond u = 0 \rightarrow \Diamond v = 0$, the direct translation yields $\tau_2(Q) = ((\mu X.u = 0 \vee \circ(X \; \mathsf{atnext} \; sch_2)) \rightarrow (\mu Y.v = 0 \vee \circ(Y \; \mathsf{atnext} \; sch_2)))$ atnext sch_2. This formula implies $\Diamond(u = 0 \wedge sch_2) \rightarrow (\mu Y.v = 0 \vee \circ(Y \; \mathsf{atnext} \; sch_2))$ which, however, still contains an alternation of minimum and maximum fixpoint formulas in its consequent, reflecting the second problem mentioned in the discussion of example 6.3.

Lemma 6.5 *Let $M = M_1 \parallel \ldots \parallel M_n$ be a composed reactive system, and let F be a closed and first-order closed PNF-formula over the language $\mathcal{L}_j$. Then* *for every* $\sigma \in \Sigma_j^\omega$ *and* $\rho \in \hat{\Sigma}^\omega$ *such that* $\rho{\downarrow_j} \preceq \sigma$ *it holds that*

$$\sigma \models F \quad \Longrightarrow \quad \rho \models \tau_j(F).$$

Proof: Let $\rho = (r_0, r_1, \ldots) \in \hat{\Sigma}^\omega$ and $\sigma = (s_0, s_1, \ldots) \in \Sigma_j^\omega$ be as in the assertion. Let the M_j-projection of ρ be $\rho{\downarrow_j} = (r_{n_0}{\downarrow_j}, r_{n_1}{\downarrow_j}, \ldots) \in \Sigma_j^* \cup \Sigma_j^\omega$. Define $N := |\rho{\downarrow_j}| \in I\!N \cup \{\omega\}$. We prove more generally that for every $\mathcal{L}_j$-formula F in positive normal form, every $0 \leq i < N$ and all valuations η and ξ of the rigid subject and the proposition variables, respectively, it holds that

$$(*) \quad \sigma_i^{\eta,\xi} \models F \quad \Longrightarrow \quad \rho_{n_i}^{\eta,\xi'} \models \tau_j^*(F)$$

where ξ' is a valuation of the proposition variables satisfying $n_l \in \xi'(X)$ iff $l \in \xi(X)$ for every $0 \leq l < N$. The assertion of the lemma follows by the assumption that F is closed and quantifier closed: If $r_l(sch_j) = \emptyset$ for every $l \geq 0$ then $\rho \models \tau_j(F)$ trivially holds. Else, for the choice of $i = 0$, by $(*)$ it follows that $\rho_{n_0} \models \tau_j^*(F)$ (for arbitrary valuations), hence the definition of n_0 as the smallest index l satisfying $r_l(sch_j) = \{()\}$ yields again $\rho \models \tau_j(F)$. The proof of $(*)$ runs by induction on the formula F:

- $F \equiv L$ is a literal: If $\sigma_i^{\eta,\xi} \models L$ where $i < N$, then $\rho_{n_i}^{\eta,\xi'} \models L \equiv \tau_j^*(L)$ because F is a $\mathcal{L}_j$-formula and $r_{n_i}{\downarrow_j} = s_i$ by the assumption $\rho{\downarrow_j} \preceq \sigma$ (and because the truth of L is independent of the valuation ξ or ξ').

- $F \equiv X$ is a proposition variable: If $\sigma_i^{\eta,\xi} \models X$ then $i \in \xi(X)$, hence $n_i \in \xi'(X)$ (by the assumption concerning ξ'), implying $\rho_{n_i}^{\eta,\xi'} \models X$.

- $F \equiv G\ op\ H$ for $op \in \{\wedge, \vee\}$, $F \equiv Qy\ G$ for $Q \in \{\forall, \exists\}$: In these cases the assertion immediately follows from the induction hypothesis.

- $F \equiv \circ G$: If $\sigma_i^{\eta,\xi} \models \circ G$ then $\sigma_{i+1}^{\eta,\xi} \models G$. If $i+1 < N$ then by induction hypothesis it follows that $\rho_{n_{i+1}}^{\eta,\xi'} \models \tau_j^*(G)$, hence $\rho_{n_i}^{\eta,\xi'} \models \circ(\tau_j^*(G)\ \mathsf{atnext}\ sch_j)$ by definition of n_{i+1}. Else, $r_l(sch_j) = \emptyset$ for every $l > n_i$ which again (trivially) implies $\rho_{n_i}^{\eta,\xi'} \models \circ(\tau_j^*(G)\ \mathsf{atnext}\ sch_j)$.

- $F \equiv \nu X.G$: Let more generally $I \subseteq I\!N$ be such that $I \subseteq \sigma^{\eta,\xi[I/X]}(G)$, and let $l \in I$ be such that $l < N$. Define $I' = \{n_p \mid p \in I \text{ and } p < N\}$ and consider any valuation ξ' satisfying the assumption $n_p \in \xi'(Y)$ iff $p \in \xi(Y)$ for every proposition variable Y and $p < N$. Then this condition also holds for the valuations $\xi'[I'/X]$ and $\xi[I/X]$, and the induction hypothesis yields $n_l \in \rho^{\eta,\xi'[I'/X]}(\tau_j^*(G))$. Since the only condition for $l \in I$ was that $l < N$, it follows that $I' \subseteq \rho^{\eta,\xi'[I'/X]}(\tau_j^*(G))$. Therefore, the assumption $i \in \sigma^{\eta,\xi}(\nu X.G) = \bigcup\{I \subseteq I\!N \mid I \subseteq \sigma^{\eta,\xi[I/X]}(G)\}$ yields $n_i \in \bigcup\{J \subseteq I\!N \mid J \subseteq \rho^{\eta,\xi'[J/X]}(\tau_j^*(G))\} = \rho^{\eta,\xi'}(\nu X.\tau_j^*(G)) = \rho^{\eta,\xi'}(\tau_j^*(\nu X.G))$.

- $F \equiv \mu X.G$: Assume that for some $i < N$, $n_i \notin \rho^{\eta,\xi'}(\mu X.\tau_j^*(G)) = \bigcap\{J \subseteq \mathbb{N} \mid \rho^{\eta,\xi'[J/X]}(\tau_j^*(G)) \subseteq J\}$ where the propositional valuation ξ' satisfies the condition $n_p \in \xi'(Y)$ iff $p \in \xi(Y)$ for every proposition variable Y and every $p < N$. Then there is some $J \subseteq \mathbb{N}$ such that $\rho^{\eta,\xi'[J/X]}(\tau_j^*(G)) \subseteq J$ and $n_i \notin J$. Define $I := \{p \in \mathbb{N} \mid n_p \in J \text{ or } p \geq N\}$. Then again it holds that $p \in \xi[I/X]$ iff $n_p \in \xi'[J/X]$ for every proposition variable Y and $p < N$. Thus, for $l < N$, if $l \in \sigma^{\eta,\xi[I/X]}(G)$ it follows by the induction hypothesis that $n_l \in \rho^{\eta,\xi'[J/X]}(\tau_j^*(G))$, implying $n_l \in J$ and, therefore, $l \in I$. If $\sigma^{\eta,\xi[I/X]} \ni l \geq N$ (assuming $N < \omega$), $l \in I$ holds anyway. Together, this implies $\sigma^{\eta,\xi[I/X]}(G) \subseteq I$ and $i \notin I$ by definition of I and the assumption $n_i \notin J$. In particular, $i \notin \bigcap\{I \subseteq \mathbb{N} \mid \sigma^{\eta,\xi[I/X]}(G) \subseteq I\} = \sigma^{\eta,\xi}(\mu X.G)$ which was to be shown. ♠

Let us now assume that for every run ρ of $M = M_1 \parallel \ldots \parallel M_n$ and every $1 \leq j \leq n$, there is some run $\sigma \in M_j^\omega$ such that $\rho{\downarrow_j} \preceq \sigma$ (a condition which will be studied in the next section). Then lemma 6.5 allows to "translate" properties of single modules into properties of the composed system which may then be strengthened using the composition axioms of lemma 6.2: If an $\mathcal{L}_j$-formula F holds in every run of the module M_j then $\tau_j(F)$ holds in every run ρ of M. Consequently, if F_j, for $1 \leq j \leq n$, expresses a property of module M_j, then the system $M = M_1 \parallel \ldots \parallel M_n$ satisfies $\tau_1(F_1) \wedge \ldots \wedge \tau_n(F_n)$.

Example 6.6 Let $M = M_1 \parallel \ldots \parallel M_n$ be a composed system and assume that (every run of) some module M_j satisfies the formula $\Box F$ where F is a state formula all of whose flexible symbols are among the module state components Π_j of M_j. In particular, F does not contain any proposition variable. By lemma 6.5 and the above assumption,

$$\tau_j(F) = (\nu X.\underbrace{F \wedge \circ(X \text{ atnext } sch_j)}_{F_X}) \text{ atnext } sch_j.$$

is true in every run ρ of M. We show in the following that even the stronger formula $\Box F$ atnext sch_j holds in every run ρ of M.

First, it follows from appropriate instances of the axioms $(P1)$ and $(P2)$ that

$$\rho \models \underbrace{\Box(F \rightarrow \circ(F \text{ unless } sch_j))}_{U}.$$

Let us now convince ourselves of the validity

$$(*) \quad \models \Box((\nu X.F_X \wedge U) \rightarrow F \wedge \circ(\nu X.F_X \wedge U))$$

from which by fixpoint induction (theorem 2.22(ii)) it follows that

$$\models \Box((\nu X.F_X \wedge U) \rightarrow \underbrace{\nu Y.F \wedge \circ Y}_{\equiv \Box F}),$$

implying the assertion $\rho \models \Box F$ atnext sch_j by $\rho \models (\nu X.F_X \wedge U)$ atnext sch_j and conventional temporal reasoning.

For the proof of $(*)$, by lemma 2.18 the outermost $\Box$-operator may be discarded. Now,

$$\begin{array}{cl}
 & \nu X.F_X \wedge U \\
\equiv & (\nu X.F \wedge \circ(\nu Y.\underbrace{(sch_j \rightarrow X) \wedge (\neg sch_j \rightarrow \circ Y)}_{F_Y})) \wedge \\
 & \nu V.\underbrace{(F \rightarrow \circ \nu W.\neg sch_j \rightarrow F \wedge \circ W) \wedge \circ V}_{F_V} \\
\overset{\text{Th.2.22(i)}}{\longleftrightarrow} & F \wedge \circ(\nu Y.(sch_j \rightarrow \nu X.F_X) \wedge (\neg sch_j \rightarrow \circ Y)) \wedge \\
 & (F \rightarrow \circ(\nu W.\underbrace{\neg sch_j \rightarrow F \wedge \circ W}_{F_W}) \wedge \circ \nu V.F_V \\
\overset{\text{Th.2.22(i),F.2.19}}{\longleftrightarrow} & F \wedge \circ(sch_j \rightarrow \nu X.F_X) \wedge \circ(\neg sch_j \rightarrow \circ \nu Y.F_Y[\nu X.F_X/X]) \wedge \\
 & \circ(\neg sch_j \rightarrow F \wedge \circ \nu W.F_W) \wedge \circ \nu V.F_V \\
\overset{\text{F.2.19}}{\longrightarrow} & F \wedge \circ(sch_j \rightarrow \nu X.F_X) \wedge \\
 & \circ(\neg sch_j \rightarrow F \wedge \circ \nu Y.F_Y[\nu X.F_X/X]) \wedge \circ U \\
\overset{\text{Th.2.22(i)}}{\longleftrightarrow} & F \wedge \circ(sch_j \rightarrow \nu X.F_X) \wedge \circ(\neg sch_j \rightarrow \nu X.F_X) \wedge \circ U \\
\overset{\text{F.2.19}}{\longleftrightarrow} & F \wedge \circ(\nu X.F_X \wedge U)
\end{array}$$

♠

The converse of lemma 6.5 does not hold in general. As an example, consider the formula f atnext sch_j which may well hold for some $\rho \in \hat{\Sigma}^\omega$ but f cannot hold in any $\sigma \in \Sigma_j^\omega$. However, the following more restricted lemma holds:

Lemma 6.7 *Let $M = M_1 \parallel \ldots \parallel M_n$ be a composed reactive system, and let F be a closed and first-order closed PNF formula over the language $\mathcal{L}_j$. Then for every $\rho \in \hat{\Sigma}^\omega$ such that $|\rho{\downarrow_j}| = \omega$ it holds that*

$$\rho \models \tau_j(F) \quad \Longrightarrow \quad \rho{\downarrow_j} \models F.$$

Proof: analogous to the proof of lemma 6.5. ♠

In particular, if $\tau_j(F)$ holds in every run of M (hence also in every run of M where M_j is scheduled infinitely often), then F holds in every run of M_j. However, this conclusion will rarely be of practical importance because properties of single modules M_j will be easier to derive from the definition of δ_j alone.

Lemma 6.5 allows properties of runs of M to be derived from properties of its constituent modules. It does not guarantee the existence of such runs (let alone the existence of runs which are extensions of arbitrary "admissible" environment runs). Moreover, properties of the single modules M_j will usually depend on the environment satisfying certain assumptions. Hence, correctness assertions of M_j will usually be of the form $U_j \rightarrow P_j$ where U_j represents the environment assumption of M_j. The translation τ_j will yield a formula of the form $\tau_j(U_j) \rightarrow \tau_j(P_j)$ which still contains (relativized versions of) the assumptions U_j. In the context of M, however, $\tau_j(U_j)$ will partly be an assertion about state components owned by the composed system, and if the modules mutually depend on each other, it will frequently not be possible to "discharge" these assumptions. In some cases one may find a layered formulation of the correctness assertions as advocated in MANNA & PNUELI [38] where the properties of the lowest layer of every module are unconditional, and the properties of higher layers depend on assumptions established in lower layers of the environment specification. In the following section, a more general approach will be investigated, identifying conditions which allow the replacement of environment assumptions for single modules by assumptions U on the environment of M. These conditions will then guarantee that arbitrary interleavings of runs of the modules M_j generate runs of the composed system M.

6.3 Discharging environment assumptions

Let us assume, as indicated at the end of the preceding section, that for every module M_j of a reactive system $M = M_1 \parallel \ldots \parallel M_n$, an environment assumption $U_j \subseteq (\Sigma_j^E)^\omega$ identifies the conditions placed on the environment of M_j to ensure that runs of M_j exist and have certain properties P_j derived from the transition condition δ_j. To ensure the existence of runs of M satisfying $\bigwedge_{j=1}^n \tau_j(P_j)$, it is necessary to guarantee fulfillment of the environment assumptions of every module. As far as the respective module's environment is also the environment of M, a suitable assumption $U \subseteq (\Sigma^E)^\omega$ will have to be found which implies (the relevant part of) U_j. For those parts of the environment of M_j formed by the other modules of M, the first idea is to stipulate

$$(P0), (P1), (P2) \models (U \wedge \bigwedge_{i \neq j} \tau_i(P_i)) \rightarrow \tau_j(U_j) \quad \text{for } 1 \leq j \leq n$$

for the derivation of $U \rightarrow \bigwedge_{j=1}^n \tau_j(P_j)$ (assuming that the properties U_i and P_i are expressed as νTL formulas), i.e. the environment assumption of M_j (relativized to runs of M) should be implied by the properties of the remaining modules (also relativized to M). However, since the modules may mutually depend on each other, this rule would in general lead to cyclic reasoning. Consider, for example, two modules M_1 and M_2 and boolean state components $p \in E_1 \cap \Pi_2, q \in E_2 \cap \Pi_1$ such that M_1 guarantees $\Box q$ provided its environment guarantees $\Diamond p$ and, similarly, M_2 guarantees $\Box p$ provided its environment satisfies $\Diamond q$. By lemma 6.5, runs of $M_1 \parallel M_2$ satisfy $\tau_1(\Diamond p \rightarrow \Box q) \wedge \tau_2(\Diamond q \rightarrow \Box p) \equiv (\tau_1(\Diamond p) \rightarrow \tau_1(\Box q)) \wedge (\tau_2(\Diamond q) \rightarrow \tau_2(\Box p))$. Assuming weak fairness for both modules, both $\tau_1(\Box q) \rightarrow \tau_2(\Diamond q)$ and $\tau_2(\Box p) \rightarrow \tau_1(\Diamond p)$ are valid. May we deduce, using the above rule, that fair runs of $M_1 \parallel M_2$ satisfy $\tau_2(\Box p) \wedge \tau_1(\Box q)$? Clearly, this is not the case: Both $\tau_1(\Diamond p \rightarrow \Box q)$ and $\tau_2(\Diamond q \rightarrow \Box p)$ are true in a run where neither p nor q ever become true.

To turn this seemingly absurd way of establishing properties of composed systems into a valid rule, the relationship between environment assumptions and module properties has to inspected somewhat more closely. In particular, partial runs instead of complete runs have to be considered.

Let us now state the relevant conditions for the general model introduced in section 6.1. In the following section, they will be specialized for νTL programs. First, let us somewhat generalize the notion of extension defined for temporal structures in chapters 4 and 5: Let $\sigma \in (\Sigma^E)^* \cup (\Sigma^E)^\omega$ be a finite or infinite state sequence over the environment state space of M. Then $\rho \in \hat{\Sigma}^* \cup \hat{\Sigma}^\omega$ is called an *extension* of σ (denoted by $\rho \sqsupseteq \sigma$ or $\sigma \sqsubseteq \rho$) if $|\rho| = |\sigma|$ and, for $0 \leq i < |\rho|$, $\rho(\lambda) = \sigma(\lambda)$ for every environment state component $\lambda \in E$. In this case, σ is also called the *restriction* of ρ to Σ^E which will be denoted as $\sigma = \rho^E$. Similar notions are defined for the state spaces Σ_j^E and Σ_j of the single modules M_j where the extension relation will be denoted by $\sqsupseteq_j$.

Theorem 6.8 *Let $M = M_1 \parallel \ldots \parallel M_n$ be a composed system where for every module M_j a transition condition $\delta_j \subseteq \Sigma_j^* \times \Sigma_j$ has been defined, and assume there exist properties $U \subseteq (\Sigma^E)^\omega$, $U_j \subseteq (\Sigma_j^E)^\omega$ and $P_j \subseteq \Sigma_j^\omega$ (for $1 \leq j \leq n$) such that the following conditions are met:*

(i) *For every $\sigma \in (\Sigma_j^E)^*$: If $\sigma \in Pref(U_j)$ then there is some extension $\sigma \sqsubseteq_j \rho \in \Sigma_j^*$ of σ such that $(\rho[..i-1], \rho[i]) \in \delta_j$ for every $0 \leq i < |\rho|$.*

(ii) *For every $\sigma \in (\Sigma_j^E)^*$: If $\sigma \in Pref(U_j)$ then for every extension $\sigma \sqsubseteq_j \rho \in \Sigma_j^*$ of σ satisfying $(\rho[..i-1], \rho[i]) \in \delta_j$ for every $0 \leq i < |\rho|$, $\rho \in Pref(P_j)$.*

(iii) *For $1 \leq j \leq n$, $U_j \neq \emptyset$.*

(iv) *For every partial run $\rho \in \hat{\Sigma}^*$ of M and $1 \leq j \leq n$: If $\rho^E \in Pref(U)$ and, for $i \neq j$, $\rho{\downarrow_i} \in Pref(P_i)$ and $(\rho{\downarrow_i})^{E_i} \in Pref(U_i)$, then $(\rho{\downarrow_j})^{E_j} \in Pref(U_j)$.*

Then for every $\sigma \in U$ and every schedule $sch : \mathbb{N} \rightarrow \{0, 1, \ldots, n\}$ there is a run $\sigma \sqsubseteq \rho \in \hat{\Sigma}^\omega$ of M such that sch is the generating schedule of ρ.

Proof: We show the following assertion: For every $\sigma \in (\Sigma^E)^*$ such that $\sigma \in \mathit{Pref}(U)$ and every schedule $sch : I\!N \to \{0, 1, \ldots, n\}$ there is an extension $\sigma \sqsubseteq \rho \in \hat{\Sigma}^*$ such that ρ is a partial run of M according to definition of 6.1, sch generates ρ, and, for every $1 \leq j \leq n$, $(\rho{\downarrow_j})^{E_j} \in \mathit{Pref}(U_j)$.

In particular, if $\sigma' \in (\Sigma^E)^\omega$ is an infinite environment state sequence such that $\sigma' \in U$ and sch is a schedule, then every prefix $\sigma \prec \sigma'$ is contained in $\mathit{Pref}(U)$, hence there exists a partial run of M generated by sch which is an extension of σ. Since the property of being a run of M is a safety property over $\hat{\Sigma}$, it follows that there exists an extension $\sigma' \sqsubseteq \rho' \in \hat{\Sigma}^\omega$ which is a run of M.

The proof of the above assertion runs by induction on the length $|\sigma| = m$ of $\sigma \in (\Sigma^E)^*$. Assume an arbitrary schedule $sch : I\!N \to \{0, 1, \ldots, n\}$.

<u>Induction Base ($m = 0$):</u> By assumption (iii), $\varepsilon \in \mathit{Pref}(U_j)$. Moreover, ε is trivially a partial run of M generated by sch. Hence, $\rho := \varepsilon$ satisfies the assertion.

<u>Induction Step:</u> Let $\sigma = (s_0, \ldots, s_m) \in (\Sigma^E)^+$ for some $m \geq 0$ and assume $\sigma \in \mathit{Pref}(U)$. Then $\sigma[..m-1] \in \mathit{Pref}(U)$, and by the induction hypothesis there is a partial run $\sigma[..m-1] \sqsubseteq \rho \in \hat{\Sigma}^*$ of M generated by sch and satisfying $\rho{\downarrow_j} \in \mathit{Pref}(U_j)$ for $1 \leq j \leq n$.

If $sch(m) = 0$ then define the state $r \in \hat{\Sigma}$ by $r(\lambda) = s_m(\lambda)$ for $\lambda \in E$, $r(\lambda) = \rho[m-1](\lambda)$ for $\lambda \in \Pi$ and $r(sch_j) = \emptyset$ for $1 \leq j \leq n$. Clearly, $\sigma \sqsubseteq \rho r$, ρr is a partial run of M generated by sch, and $(\rho r){\downarrow_i} = \rho{\downarrow_i} \in \mathit{Pref}(U_i)$ for $1 \leq i \leq n$ by the induction hypothesis.

Else, assume that $sch(m) = j \in \{1, \ldots, n\}$. For every state $r' \in \hat{\Sigma}$ satisfying

$$r'(\lambda) = \left\{ \begin{array}{cl} s_m(\lambda) & \text{if } \lambda \in E, \\ \rho[m-1](\lambda) & \text{if } \lambda \in \Pi_i, i \neq j, \\ \emptyset & \text{if } \lambda \equiv sch_i, i \neq j, \\ \{()\} & \text{if } \lambda \equiv sch_j \end{array} \right\}$$

(and assigning arbitrary values $r'(\lambda)$ to state components $\lambda \in \Pi_j$) it follows that $((\rho r'){\downarrow_i})^{E_i} = (\rho{\downarrow_i})^{E_i} \in \mathit{Pref}(U_i)$ for every $i \neq j$. Since ρ is a partial run of M and $(\rho r'){\downarrow_i} = \rho{\downarrow_i}$ for every $i \neq j$, the assumption (ii) implies that for every such i, $(\rho r'){\downarrow_i} \in \mathit{Pref}(P_i)$. Moreover, $(\rho r')^E = \sigma \in \mathit{Pref}(U)$, thus by assumption (iv), $((\rho r'){\downarrow_j})^{E_j} \in \mathit{Pref}(U_j)$. Therefore, assumption (i) implies that there is some state $r'' \in \Sigma_j$ such that $r''(\lambda) = r'(\lambda)$ for every $\lambda \in E_j$ and $(\rho{\downarrow_j}, r'') \in \delta_j$. Now define $r \in \hat{\Sigma}$ by $r(\lambda) = r''(\lambda)$ for $\lambda \in \Pi_j$ and $r(\lambda) = r'(\lambda)$ else (observe that this is possible since $\Pi_i \cap \Pi_j = \emptyset$ for $i \neq j$. Then $r{\downarrow_j} = r''$ which, together with the induction hypothesis and $(\rho{\downarrow_j}, r'') \in \delta_j$, implies that $\rho r \sqsupseteq \sigma$ is a partial run of M generated by the schedule sch and satisfying $(\rho r){\downarrow_i} \in \mathit{Pref}(U_i)$ for $1 \leq i \leq n$. ♠

Discussion of the hypotheses of theorem 6.8

Assumption (i) requires, for every module M_j, that the existence of a partial run ρ of M_j as an extension of a given environment structure σ only depends on the finite prefix of σ corresponding to ρ. This requirement fits well with our general stipulation in section 4.1 that runs of reactive systems be constructed incrementally. In fact, it will be seen to be automatically satisfied by the usage of environment assumptions for νTL programs in chapters 4 and 5. It cannot be satisfied by an environment assumption like $\Diamond p$ used in the discussion before theorem 6.8. In fact, viewing hypothesis (i) as a requirement for $U_j \subseteq (\Sigma_j^E)^\omega$, it is formulated as a safety property. Applying lemma 2.11, it follows that U_j may be replaced by its safety closure to establish hypothesis (i). Similar remarks apply to the second hypothesis.

The third hypothesis requires the environment assumptions of every single module to be non-empty (i.e. to be expressed by satisfiable formulas when theorem 6.8 is translated to a proof rule for program verification). Since, in general, every environment assumption U_j is partly a correctness assertion for the other modules, this is clearly necessary for the existence of runs, assuming that U is non-empty. Finally, the fourth hypothesis is, by lemma 6.5, a somewhat refined version of the naive proof rule suggested at the beginning of this section. Similarly as in the formulation of the first two hypotheses, the condition is required to hold for partial runs.

It remains to be seen how the assumptions (i)–(iv) of theorem 6.8 can be established for modules defined in a particular formalism. For νTL programs, this will be studied in the following section.

6.4 Composing νTL programs

Assume the module M_j to be given as a (propositional or first-order) νTL program $\Phi_j = \{\varphi^1, \dots, \varphi^q\}$, and let $T^1, \dots, T^q$ denote the transition graphs (or schemes) associated with $\varphi^1, \dots, \varphi^q$. As in chapter 5, assume a fixed first-order structure $\mathcal{S}$ in the case of first-order programs. The operational semantics of Φ_j given by algorithm 4.32 or 5.32, respectively, can be represented in the more general framework discussed in this chapter by defining a transition condition $\delta_j \subseteq \Sigma_j^* \times \Sigma_j$ as follows: Let $\sigma = (s_0, \dots, s_{k-1}) \in \Sigma_j^*$ be a finite (possibly empty) state sequence, and $s \in \Sigma_j$. Denote by $M^i = (m_0^i, \dots, m_k^i)$, for $1 \leq i \leq q$, the marking of T^i over the finite state sequence σs (defined analogously as in definitions 4.8 or 5.9 for infinite state sequences). If Φ_j is a propositional νTL program, $(\sigma, s) \in \delta_j$ iff for every $1 \leq i \leq q$ and $n \in m_k^i$, $s(p) = \{()\}$ for every $p \in \vartheta^i(n)$ and $s(p) = \emptyset$ for every $p \in \mathcal{V}$ such that $\neg p \in \vartheta^i(n)$. In the first-order case, the relevant conditions are that $(d_1, \ldots, d_m) \in s(P)$ (resp., $s(z) = d$) iff there is some node $n \in N^i$ (for some $1 \leq i \leq q$), some formula $P(t_1, \ldots, t_m) \in \vartheta^i(n)$ (resp., $z = t \in \vartheta^i(n)$) and some valuation $\eta : \mathcal{Y} \to |\mathcal{S}|$ satisfying $\eta(V^i(n)) \in m^i_k(n)$ such that $s^\eta(t_1) = d_1, \ldots, s^\eta(t_m) = d_m$ (resp., $s^\eta(t) = d$) where the evaluation of terms over a single state and a valuation of the rigid variables is defined in the canonical way.

As a consequence of this definition and lemma 4.11, or lemma 5.12 and definition 5.25, a state sequence (i.e. temporal structure) $\sigma \in \Sigma_j^\omega$ is a run of M_j in the sense of definition 6.1 iff it is a model (resp., a normal model in the first-order case) of Φ_j.

Let us now turn to the existence of runs of composed systems. For a single νTL program Φ_j, the compatibility conditions of definitions 4.31 and 5.31, respectively, guaranteed the existence of (normal) models, i.e. runs of Φ_j. These were defined relative to environment assumptions $\mathcal{U}_j$, the relevant condition being given by the acceptance of certain temporal structures by the conflict graphs (or schemes) corresponding to all pairs of the individual program clauses. Since every transition graph (or scheme) defines a safety property, this condition is again a safety property: A structure σ is accepted by any transition graph (or scheme) T iff every finite prefix of σ can be extended to a structure σ' accepted by T. This will automatically establish the first two assumptions of theorem 6.8:

Lemma 6.9 *Let Φ be a $\mathcal{U}$-compatible propositional or first-order νTL program where Σ, Σ^E and Σ^Π denote the state space of Φ (resp., the restrictions of Σ to the environment-owned and module-owned state components), and $\mathcal{U} \subseteq (\Sigma^E)^\omega$. Then, for every finite state sequence $\sigma \in (\Sigma^E)^*$ such that $\sigma \in Pref(\mathcal{U})$ there is an extension $\sigma \sqsubseteq \rho \in \Sigma^*$ such that ρ is a partial run of Φ, and every such ρ is a prefix of a model (resp., of a normal model in the first-order case) of Φ.*

Proof: If $\sigma \in Pref(\mathcal{U})$, then there is some $\sigma' \in (\Sigma^E)^\omega$ such that $\sigma\sigma' \in \mathcal{U}$. Hence, theorems 4.33 or 5.33, respectively, imply that there is a run $\sigma\sigma' \sqsubseteq \rho' \in \Sigma^\omega$ of Φ. Setting $\rho := \rho'[..|\sigma| - 1]$ validates the first claim of the lemma. On the other hand, differences in the valuations of state components $\lambda \in \Pi$ in two (normal) models, i.e. partial runs $\rho \sqsupseteq \sigma$ of Φ, have to be caused by different assignments to "undefined" state components (in the sense of steps (1a) and 2 of the respective interpreter algorithms), hence they are possible outcomes of these algorithms and can thus, by theorems 4.33 and 5.33, be completed to (normal) models of Φ. ♠

Lemma 6.9 implies that for modules M_j given as $\mathcal{U}_j$-compatible νTL programs Φ_j, only the conditions (iii) and (iv) of theorem 6.8 have to be established. Assume now that the environment assumption $\mathcal{U}_j$ of every module M_j is expressed by a closed and first-order closed νTL formula Ψ_j (where only environment components $\lambda \in E_j$ appear in Ψ_j), and that the environment condition of $M_1 \parallel \ldots \parallel M_n$ is expressed by the formula Ψ. Then condition (iii) amounts to showing the satisfiability of every Ψ_j. Condition (iv), on the other hand, can be restated as a set of νTL formulas which have to be derived using the axioms $(P0)-(P2)$ of lemma 6.2 to guarantee the existence of runs of the composed system M. In the sequel, all temporal structures will be sequences in $\hat{\Sigma}^\omega$ (hence, all formulas of the different sublanguages of $\hat{\mathcal{L}}$ can be interpreted in those structures).

Lemma 6.10 *Let $M = M_1 \parallel \ldots \parallel M_n$ be a reactive system whose modules M_j are given as $\mathcal{U}_j$-compatible νTL programs Φ_j where $\mathcal{U}_j = Mod(\Psi_j)$ for some νTL formula Ψ_j (over the language E_j). Let $\mathcal{U} = Mod(\Psi)$ (where Ψ is a νTL formula over the language E) be an environment assumption for M. Let Φ_j^c denote Φ_j if Φ_j is a propositional νTL program, and the completion of Φ_j (see definition 5.29) else. Assume F_j, for $1 \leq j \leq n$, to be consequences of Φ_j^c and Ψ_j such that for $1 \leq i \leq n$, the following formulas follow from appropriate instances of $(P0)$, $(P1)$ and $(P2)$:*

$$(\mathrm{COMP})_i \quad (\Psi \wedge \bigwedge_{j \neq i} \tau_j(F_j)) \rightarrow \tau_i(\Psi_i).$$

Then for every $1 \leq i \leq n$ and every partial run $\rho \in \hat{\Sigma}^*$ of M such that $\rho^E \in Pref(\mathcal{U})$ and $(\rho{\downarrow_j})^{E_j} \in Pref(\mathcal{U}_j)$ for every $j \neq i$, $(\rho{\downarrow_i})^{E_i} \in Pref(\mathcal{U}_i)$.

Proof: Let ρ be a partial run of M as in the assertion. Then there are state sequences $\rho' \in \hat{\Sigma}^\omega$, $\rho''_j \in (\Sigma_j^{E_j})^\omega$ such that $\rho\rho' \models \Psi$ and $(\rho{\downarrow_j})^{E_j}\rho''_j \models \Psi_j$ for every $j \neq i$. Moreover, every $\rho{\downarrow_j}$ is a partial run of Φ_j. Because all Φ_j are $\mathcal{U}_j$-compatible programs, by lemma 6.9 there exist $\rho''_j \sqsubseteq \rho'_j \in \hat{\Sigma}^\omega$ such that $\rho{\downarrow_j}\rho'_j \models \Psi_j \wedge \Phi_j^c$, hence also $\rho{\downarrow_j}\rho'_j \models F_j$.

Since Ψ does not contain any of the proposition variables sch_j nor any state components $\lambda \in \Pi_j$, assume w.l.o.g. that, for every $l \geq 0$, $\rho'[l](sch_j) = \emptyset$ for $j \neq i$, $\rho'[l](sch_i) = \{()\}$ and $\rho'[l](\lambda) = \rho[|\rho|-1](\lambda)$ for every $\lambda \in \bigcup_{j \neq i} \Pi_j$. Then $(\rho\rho'){\downarrow_j} = \rho{\downarrow_j} \preceq \rho{\downarrow_j}\rho'_j$ for every $j \neq i$, hence by lemma 6.5 it follows that $\rho\rho' \models \tau_j(F_j)$ for every $j \neq i$. Clearly, by the assumption that ρ is a partial run of M and the definition of ρ', $\rho\rho'$ is a model of the axioms $(P0)$, $(P1)$ and $(P2)$. Hence, $(\mathrm{COMP})_i$ holds in $\rho\rho'$, and by $\rho\rho' \models \Psi$ and $\rho\rho' \models \tau_j(F_j)$ for every $j \neq i$, it follows that $\rho\rho' \models \tau_i(\Psi_i)$. Now, $|(\rho\rho'){\downarrow_i}| = \omega$ by definition of ρ', hence lemma 6.7 implies that $(\rho\rho'){\downarrow_i} \models \Psi_i$, in particular $(\rho{\downarrow_i})^{E_i} \in Pref(\mathcal{U}_i)$. ♠

The surprising result of theorem 6.8, in combination with the lemmas 6.9 and 6.10, is therefore that the naive composition rule given at the beginning of section 6.3 is sound for νTL programs provided that the environment assumptions of the single modules are consistent. A similar rule (although without an accompanying proof) has appeared in Abadi & Lamport [3] in a different formal framework where the authors similarly pointed out that the main requirement for the soundness of the rule was that all environment assumptions were safety properties.

An example for an application of this composition rule for the verification of a parallel system is provided in chapter 7. For practical purposes, the rule is still a bit weak in one respect: The modules have to cope with completely arbitrary initial values of state components owned by other modules of the system. Formally, this is reflected by the syntactical form of the formulas $\tau_j(F_j) \equiv \tau_j^*(F_j)$ atnext sch_j expressing properties of the composed system which allow nothing to be derived about the states before M_j is scheduled for the first time; also, the axioms $(P0)$–$(P2)$ are of no help here. The practical consequence of this deficiency is that for every system a protocol has to be devised ensuring that all modules have been initialized and know about each other's initialization so that they can rely on the values "received" from other components to be valid outputs conforming to the environment assumption made about these values. Such a protocol may be quite hard to develop besides obscuring the programs implementing the single modules' specifications. It is, however, reasonable to assume that such initialization problems may be left to the "operating system" (or interpreter). Let us conclude this chapter by sketching a solution of this problem.

Let Φ_j be a νTL program implementing module M_j of the composed system $M = M_1 \parallel \ldots \parallel M_n$ whose set of module state components is Π_j, and let $\Pi_j^0 \subseteq \Pi_j$ be a set of *initialized state components*. An *initialization formula* for Φ_j is of the form Θ_j unless sch_j where

$$\Theta_j \equiv \bigwedge_{p \in \Pi_j^0} (\neg) p,$$

i.e. a conjunction of possibly negated proposition letters contained in Π_j^0 if Φ is a propositional νTL program, and

$$\Theta_j \equiv \bigwedge_{z \in \Pi_j^0 \cap \mathcal{Z}} z = t \quad \wedge \bigwedge_{P \in \Pi_j^0 \cap \mathcal{P}} \forall \vec{x}\, (P(\vec{x}) \leftrightarrow \bigvee_{i=1}^{m_P} \vec{x} = \vec{t}_i)$$

where $\vec{x}$ and $\vec{t}_i$ denote tuples of variables and terms, respectively, whose size agrees with the arity of P, and all terms are built from constants and function applications only, if Φ_j is a first-order νTL program. Such initialization formulas assign an initial value to module-owned flexible variables or a finite extension to module-owned predicates and are clearly realizable by a suitable operating system or interpreter mechanism since they only restrict the interpretation of state components owned by module M_j which are not written to by the other modules. Call $\sigma \in \hat{\Sigma}^\omega$ an *initialized run* of M if it is a run of M and if, moreover, $\sigma \models \Theta_j$ for every $1 \leq j \leq n$ such that $\sigma[0](sch_j) = \emptyset$. (The definition of runs of M ensures that this implies $\sigma \models \Theta_j$ unless sch_j for every $1 \leq j \leq n$). Now the composition rule for $M = M_1 \parallel \ldots \parallel M_n$ may be given in the form

$$\begin{array}{|l|}
\hline
\Psi_j \text{ is satisfiable for } 1 \leq j \leq n \\
\Psi_j, \Phi_j^c \;\models\; F_j \text{ for } 1 \leq j \leq n \\
(P0), (P1), (P2), \Psi, \{\Theta_j \;\mathsf{unless}\; sch_j, \tau_j(F_j) \mid j \neq i\} \;\models\; \tau_i(\Psi_i) \text{ for } 1 \leq i \leq n \\
\hline
M \text{ has a run } \sigma \text{ and } \sigma \models \bigwedge_{j=1}^{n} \tau_j(F_j) \text{ for every initialized run } \sigma \text{ of } M \\
\hline
\end{array}$$

This rule will be used in the verification of an example system consisting of two modules given as νTL programs in chapter 7.

Chapter 7

Examples of application

The development of reactive systems using νTL as a specification and implementation language will be illustrated by studying two examples which hint at how more realistic problems could be addressed. The examples chosen are of toy size to facilitate comparison with other existing development and programming frameworks and allow a reasonably complete presentation. The development of larger systems would clearly need machine assistance; in particular, proofs required during the development process are often tedious — although usually not very difficult — to carry out manually. Our first example demonstrates the development of a single module as a νTL program, the second example is concerned with the composition of modules and the derivation of system properties when the correctness of the single modules depends on the correctness of each other.

7.1 Dining philosophers

The first example deals with one of the best-known problems in the theory of distributed reactive systems which originally goes back to DIJKSTRA and allows the study of most properties of interest in typical reactive systems, such as mutual exclusion and absence of deadlock and starvation, in a nutshell-size system: Consider $n \geq 2$ philosophers living in a monastery and sharing a round table in the refectory. The philosophers do not obey any fixed schedule with respect to their working, sleeping or eating hours. Whenever a philosopher gets hungry, he goes to the common table in the middle of which there is a bowl with a never-ending supply of spaghetti. Around the bowl there are n plates — each philosopher has his own plate — and n forks, one to the left and one to the right of each plate. Every philosopher needs the forks on both sides of his plate, hence he cannot eat as long as one of his neighbours is eating.

The problem is to develop a system ensuring that every philosopher getting hungry will eat eventually (obviously it has to be required that no philosopher eats forever). We will give a centralized solution to this problem by developing a "butler" module which administers the forks, keeping a record of which philosophers are hungry and ensuring that every hungry philosopher eventually gets a turn at the table. The philosophers tell the butler when they are hungry and inform him when they have finished eating. A philosopher may (and will be assumed to) start eating as soon as he has been given both forks. No forks may be reclaimed from a philosopher owning both forks unless the philosopher has finished eating and has informed the butler about this.

The specification of the butler module uses the environment predicates (as seen by the butler) $\mathrm{Hungry}(x)$ and $\mathrm{Enough}(x)$ as well as the module predicate $\mathrm{Eat}(x)$ with their obvious meanings. To fix the first-order kernel of both the language and structures over which the following formulas are interpreted, assume a constant 0 and functions $\oplus$ and $\ominus$ (applied in infix notation) which are interpreted by the number zero and addition and subtraction modulo n, respectively. (In particular, the universe $|\mathcal{S}|$ contains the set $\{0, \ldots, n-1\} \subseteq \mathbb{N}$ to identify the philosophers). For any philosopher x, $x \oplus 1$ and $x \ominus 1$ denote, respectively, the right and left neighbour of x. A specification of the butler's behaviour is given by

(7.1) $\forall x\, \Box(\mathrm{Hungry}(x) \rightarrow \Diamond \mathrm{Eat}(x))$

(7.2) $\forall x\, \Box\neg(\mathrm{Eat}(x) \wedge \mathrm{Eat}(x \oplus 1))$

(7.3) $\forall x\, \Box(\mathrm{Eat}(x) \rightarrow \mathrm{Eat}(x) \ \mathsf{unless}\ \mathrm{Enough}(x))$

(7.4) $\forall x\, \neg\mathrm{Eat}(x)$

The formulas (7.1) – (7.3) reflect the behaviour informally described above. Formula (7.4) introduces the additional assumption that initially no philosopher is eating. Observe that the forks, although prominent in the informal description to motivate the desired behaviour of the system, do not appear in this formal specification.

The principal development effort has to be spent on implementing (7.1). This is reflected in the fact that (7.1) is the only non-safety formula of the specification. First, the butler records which philosophers are waiting to be served. (Alternatively, the philosophers could be required to uphold the condition $\mathrm{Hungry}(x)$ until they are being served.) A new module predicate Waiting is introduced together with the program clause

(7.5) $\Box(\mathrm{Hungry}(x) \rightarrow \mathrm{Waiting}(x)\ \mathsf{unless}\ \mathrm{Eat}(x) \vee \mathrm{Enough}(x))$.

Let us, for every newly introduced program clause, also record the corresponding requirements on the dependency order of the module state components for the respective clause to be syntactically admissible according to definition 5.3. Unabbreviated, clause (7.5) corresponds to $\nu X.(\mathrm{Hungry}(x) \rightarrow \nu Y.\neg(\mathrm{Eat}(x) \vee \mathrm{Enough}(x)) \rightarrow \mathrm{Waiting}(x) \wedge \circ Y) \wedge \circ X$, hence Eat < Waiting has to be stipulated. (Observe that Hungry < Waiting and Enough < Waiting hold anyway since Hungry and Enough are environment components.)

Next, the butler has to assign forks to waiting philosophers and keep track of which forks are currently available at any given moment. Let us introduce two predicates Left(x) and Right(x) indicating that philosopher x has been assigned his left or right fork, respectively. Observe that this is just one possible approach which has the drawback that every fork is represented twice, i.e., the butler has to be careful never to allow Left(x) and Right($x \ominus 1$) to hold at the same time. The first idea is to give a waiting philosopher his left fork as soon as it is not in use as his left neighbour's right fork, i.e.,

(7.6) $\Box(\mathrm{Waiting}(x) \rightarrow \mathrm{Left(x)}\ \mathsf{atnext}\ \neg\mathrm{Right(x} \ominus 1)).$

Clause (7.6) requires Waiting < Left and Right < Left to hold w.r.t. the dependency order.

A philosopher holding his left fork may similarly be assigned his right fork as soon as it is free, i.e., when it is not in use by his right neigbour. From this moment, he may eat and, in particular, keep the forks until he finishes eating. This is expressed by the following program clause

(7.7) $\Box(\mathrm{Left(x)} \wedge \neg\mathrm{Left(x} \oplus 1) \rightarrow \circ(\mathrm{Right(x)} \wedge \mathrm{Eat(x)}\ \mathsf{unless}\ \mathrm{Enough(x)}))$

Observe that the $\circ$-operator in the consequent part is necessary in order not to violate the requirement Right < Left introduced above.

However, the program is not yet complete: Clause (7.6) only assigns a philosopher his left fork for one state without stating for how long he may keep it. The most straightforward modification of clause (7.6) would be

$$\Box(\mathrm{Waiting}(x) \rightarrow (\mathrm{Left(x)}\ \mathsf{unless}\ \mathrm{Enough(x)})\ \mathsf{atnext}\ \neg\mathrm{Right(x} \ominus 1)),$$

i.e., to allow a philosopher to keep his left fork until he finishes eating and signals that he does not need it anymore. However, the resulting program would not be deadlock-free (formally, the formula (7.1) of the original specification would not follow from the program obtained in this way). For example, if all philosophers got hungry simultaneously, all would immediately be assigned their left forks by the modified clause (7.6), but no philosopher could be given his right fork. We resolve this problem as follows: One specific philosopher is preferred by the butler in the following way: If this philosopher *pref* and his right neighbour $pref \oplus 1$ both hold their respective left forks but not the right ones then the butler reclaims the left fork of philosopher $pref \oplus 1$ and assigns it to philosopher $pref$ as his right fork — but then this philosopher loses his preferred status to his right neighbour who has been "discriminated" against. Observe that this impolite behaviour of the butler is allowed by the original specification; specifically, (7.3) only applies if the philosopher holds both forks. Formally, a flexible variable $pref$ is introduced indicating the philosopher currently enjoying the butler's preference. Let Reclaim(x) be an abbreviation for the state formula

$$\mathrm{Reclaim}(x) \;\equiv\; pref = x \wedge \mathrm{Left(x)} \wedge \mathrm{Left(x \oplus 1)} \wedge \neg\mathrm{Right(x \oplus 1)}$$

describing the conflict situation. We introduce the additional program clauses

$$
\begin{array}{ll}
(7.8) & pref = 0 \;\mathsf{upto}\; \mathrm{Reclaim}(0) \\
(7.9) & \Box(\mathrm{Reclaim}(x) \rightarrow \circ((\mathrm{Right}(x) \wedge \mathrm{Eat}(x) \;\mathsf{unless}\; \mathrm{Enough}(x)) \wedge \\
& \qquad\qquad\qquad (pref = x \oplus 1 \;\mathsf{upto}\; \mathrm{Reclaim}(x \oplus 1))))
\end{array}
$$

where the temporal operator upto is defined, as in example 4.34, by

$$
\begin{array}{rcl}
F \;\mathsf{upto}\; G & \equiv & \nu X.F \wedge (\neg G \rightarrow \circ X) \\
& \equiv & (F \;\mathsf{unless}\; G) \wedge (F \;\mathsf{atnext}\; G).
\end{array}
$$

The clauses (7.8) and (7.9) do not introduce new requirements for the order of the module state components.

Now, clause (7.6) is replaced by

$$
\begin{array}{ll}
(7.10) & \Box(\mathrm{Waiting}(x) \rightarrow (\mathrm{Left(x)} \;\mathsf{upto}\; \mathrm{Reclaim(x \ominus 1)} \vee \mathrm{Enough(x)}) \\
& \qquad\qquad\qquad \mathsf{atnext}\; \neg\mathrm{Right}(x \ominus 1)).
\end{array}
$$

Observe that the completion axioms (see definition 5.29) will ensure that philosopher x actually loses his left and/or right fork when the conditions of the unless-formulas of clauses (7.7) and (7.10) are met.

The resulting νTL program implementing the behaviour of a butler for n dining philosophers is given in table 7.1.

It has to be checked that the butler program is compatible w.r.t. some reasonable environment assumption. The only possible conflicts can occur in the assignments to the variable $pref$ in clauses (7.8) and (7.9). Hence, the conflict schemes of (the regular, complete and monotonic transition schemes associated with) the pairs (7.8, 7.8), (7.8, 7.9) and (7.9, 7.9) of program clauses have to be constructed. All these conflict schemes turn out to represent valid conditions, implying that the program is $\mathcal{U}$-compatible w.r.t. any (or no) environment assumption. As an example, the (somewhat simplified) conflict scheme of (7.9)

(7.5)	$\Box(\mathrm{Hungry}(x) \rightarrow \mathrm{Waiting}(x)$ **unless** $\mathrm{Eat}(x) \vee \mathrm{Enough}(x))$
(7.10)	$\Box(\mathrm{Waiting}(x) \rightarrow (\mathrm{Left(x)}$ **upto** $\mathrm{Reclaim}(\mathrm{x} \ominus 1) \vee \mathrm{Enough(x)})$
	atnext $\neg\mathrm{Right}(x \ominus 1))$
(7.7)	$\Box(\mathrm{Left(x)} \wedge \neg\mathrm{Left}(\mathrm{x} \oplus 1) \rightarrow \circ(\mathrm{Right(x)} \wedge \mathrm{Eat(x)}$ **unless** $\mathrm{Enough(x)}))$
(7.8)	$\mathit{pref} = 0$ **upto** $\mathrm{Reclaim}(0)$
(7.9)	$\Box(\mathrm{Reclaim}(x) \rightarrow \circ((\mathrm{Right}(x) \wedge \mathrm{Eat}(x)$ **unless** $\mathrm{Enough}(x)) \wedge$
	$(\mathit{pref} = x \oplus 1$ **upto** $\mathrm{Reclaim}(x \oplus 1))))$
	where
	$\mathrm{Reclaim}(x) \equiv \mathit{pref} = x \ \wedge\ \mathrm{Left(x)} \ \wedge\ \mathrm{Left}(\mathrm{x} \oplus 1) \ \wedge\ \neg\mathrm{Right}(\mathrm{x} \oplus 1)$

Table 7.1: Butler for n philosophers

with itself is shown in figure 7.1. Recall that two copies of (7.9) using different rigid subject variables have to be used.

The conflict scheme represents the condition

$$\forall x \forall y\, \Box(\mathrm{Reclaim}(x) \ \wedge\ \mathrm{Reclaim}(y) \rightarrow \circ(x \oplus 1 = y \oplus 1 \ \textsf{upto}\ \mathrm{Reclaim}(x \oplus 1) \vee \mathrm{Reclaim}(y \oplus 1)))$$

which is valid because $\mathrm{Reclaim}(x) \wedge \mathrm{Reclaim}(y)$ implies $x = y$ (by $\mathit{pref} = x$ and $\mathit{pref} = y$), hence $x \oplus 1 = y \oplus 1$ and also $\Box(x \oplus 1 = y \oplus 1)$ (because no flexible symbols are involved), implying the weaker assertion expressed by the conflict scheme.

The correctness of the butler program remains to be proven, i.e., it has to be shown that the completion of the program listed in table 7.1 implies the original specification given by the formulas (7.1) – (7.4). Clearly, the most difficult part is the proof of (7.1). Indeed, this formula does not follow from the completion of the butler program alone, but needs additional assumptions about the behaviour of the philosophers. One such assumption which has already been mentioned above states that no philosopher eats forever, i.e., every eating philosopher eventually signals “enough”:

(7.11) $\forall x \Box(\mathrm{Eat}(x) \rightarrow \Diamond\mathrm{Enough}(x))$.

An assumption hidden in the informal presentation of the problem is that the life of every philosopher (as far as it is relevant for the butler) is a continuous

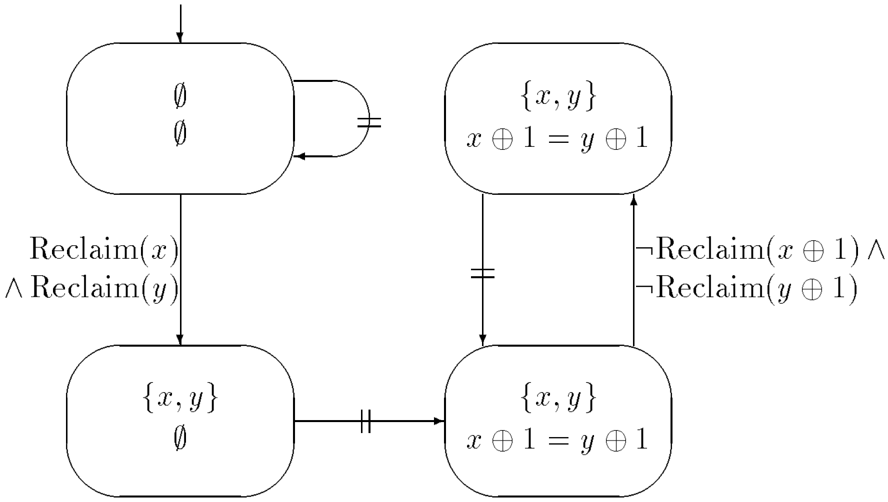


Figure 7.1: (Simplified) conflict scheme of 7.9 with itself

cycle "thinking — becoming hungry — eating", but there is no formal requirement reflecting this assumption. What happens if a philosopher x, having declared himself hungry and having waited for some time, decides to resume thinking instead of waiting longer and signals Enough(x) before being served? A quick inspection of the butler program (together with the implicit completion axioms) shows that it reacts quite reasonably, eliminating x from the list of waiting philosophers and possibly reclaiming his left fork. Clearly, in this case the formula (7.1) may be violated, but this is not the fault of the butler. Without specifying the philosophers' behaviour exactly, we will make the somewhat simplifying assumption that no philosopher simultaneously signals "hungry" and "enough", i.e., we assume

(7.12) $\forall x \Box \neg(\text{Hungry}(x) \wedge \text{Enough}(x))$.

What can now be shown is that — under the assumptions (7.11) and (7.12) — the butler module satisfies

(7.13) $\forall x \Box (\text{Hungry}(x) \rightarrow \Diamond(\text{Eat}(x) \vee \text{Enough}(x)))$.

Only the main steps of a formal derivation of this formula from the completion of the butler program and the assumptions (7.11) and (7.12) will be given. It could be carried out more formally in an axiom system for temporal logic, like those appearing in Kröger [29] or in Manna & Pnueli [38]. Derivation steps which involve reasoning about the fixed first-order structure used for this example will be marked by (data). The derivation uses the following invariants

$(I1)$–$(I7)$ which can be derived from the completion of the butler program without using the assumptions (7.11) and (7.12).

$(I1)$ $\forall x \Box(\neg \mathrm{Eat}(x) \to \neg \mathrm{Right}(x))$
$(I2)$ $\forall x \Box(\mathrm{Enough}(x) \to \neg \mathrm{Right}(x))$
$(I3)$ $\forall x \Box(\mathrm{Waiting}(x) \to \mathrm{Waiting}(x) \ \mathsf{unless}\ \mathrm{Eat}(x) \vee \mathrm{Enough}(x))$
$(I4)$ $\forall k \forall x \Box(\mathrm{Waiting}(x) \wedge x \ominus \mathit{pref} \leq k \to x \ominus \mathit{pref} \leq k \ \mathsf{unless}\ \mathrm{Eat}(x))$
$(I5)$ $\forall x \Box(\mathrm{Enough}(x) \wedge \neg \mathrm{Hungry}(x) \to \neg \mathrm{Left}(\mathrm{x}))$
$(I6)$ $\forall x \Box(\mathrm{Left}(\mathrm{x} \oplus 1) \to \neg \mathrm{Right}(\mathrm{x}))$
$(I7)$ $\forall x \Box(\mathrm{Left}(\mathrm{x}) \to \mathrm{Left}(\mathrm{x}) \ \mathsf{upto}\ \mathrm{Reclaim}(\mathrm{x} \ominus 1) \vee \mathrm{Enough}(\mathrm{x}))$

As an example, let us derive the invariant $(I7)$ from the completion of the butler program. To this end, the completion axiom $(\mathrm{DEF}_{\mathrm{Left}})$ for the predicate symbol Left has to be stated. The only clause of the program which "writes" to Left is (7.10). Its associated transition scheme is shown in figure 7.2, together with the transition axioms for this transition scheme.

Now, the completion axiom $(\mathrm{DEF}_{\mathrm{Left}})$ becomes

$$\begin{aligned}(\mathrm{DEF}_{\mathrm{Left}}) \quad &\equiv \quad \Box \forall y (\mathrm{Left}(\mathrm{y}) \leftrightarrow \exists \mathrm{x}(\mathrm{M}_3(\mathrm{x}) \wedge \mathrm{y} = \mathrm{x})) \\ &\equiv \quad \Box \forall x (\mathrm{Left}(\mathrm{x}) \leftrightarrow \mathrm{M}_3(\mathrm{x})).\end{aligned}$$

Using the transition axioms shown in figure 7.2, we conclude that

$$\forall x \Box(M_3(x) \to \mathrm{Left}(\mathrm{x}) \wedge (\neg(\mathrm{Reclaim}(\mathrm{x} \ominus 1) \vee \mathrm{Enough}(\mathrm{x})) \to \circ \mathrm{M}_3(\mathrm{x})))$$

follows from the completion of the butler program, hence by fixpoint induction (theorem 2.22(ii)) also

$$\forall x \Box(M_3(x) \to \mathrm{Left}(\mathrm{x}) \ \mathsf{upto}\ \mathrm{Reclaim}(\mathrm{x} \ominus 1) \vee \mathrm{Enough}(\mathrm{x})),$$

and, using the completion axiom $(\mathrm{DEF}_{\mathrm{Left}})$, also

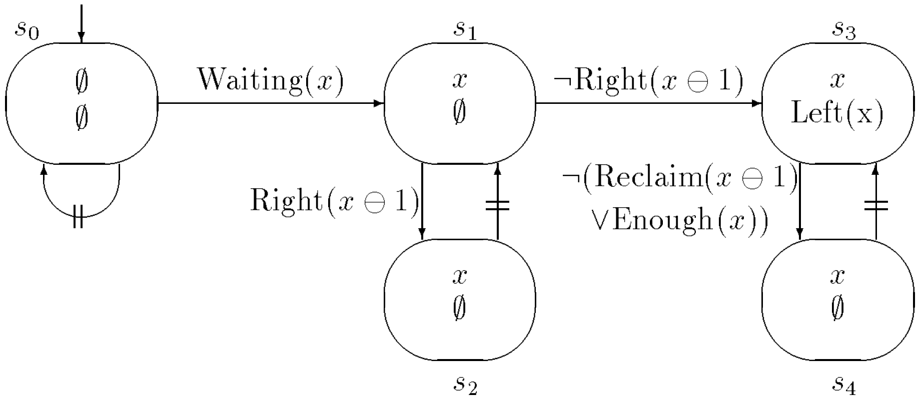
$$\forall x \Box(\mathrm{Left}(\mathrm{x}) \to \mathrm{Left}(\mathrm{x}) \ \mathsf{upto}\ \mathrm{Reclaim}(\mathrm{x} \ominus 1) \vee \mathrm{Enough}(\mathrm{x})),$$

i.e., the invariant $(I7)$.

As the principal step for the derivation of (7.13), we show that

$$\begin{aligned}(*) \quad \forall x \forall k \Box(&\mathrm{Waiting}(x) \wedge x \ominus \mathit{pref} \leq k \to \\ &\Diamond((\mathrm{Waiting}(x) \wedge x \ominus \mathit{pref} < k) \vee \mathrm{Eat}(x) \vee \mathrm{Enough}(x))\end{aligned}$$

follows from the completion of the butler program and the assumptions (7.11) and (7.12). The correctness assertion (7.13) follows from $(*)$ by well-founded induction on $(\{0, \ldots, n-1\}, <)$ and the program clause (7.5), observing that $\forall x \Box(\mathrm{Hungry}(x) \to \exists k (x \ominus \mathit{pref} \leq k))$ is valid in the fixed first-order structure for the butler program.



Transition axioms:

M_0
$\Box(\bigcirc M_0 \leftrightarrow M_0)$
$\forall x(M_1(x) \leftrightarrow M_0 \wedge \mathrm{Waiting}(x))$
$\Box\forall x(\bigcirc M_1(x) \leftrightarrow \bigcirc(M_0 \wedge \mathrm{Waiting}(x)) \vee M_2(x))$
$\forall x(M_2(x) \leftrightarrow M_1(x) \wedge \mathrm{Right}(x \ominus 1))$
$\Box\forall x(\bigcirc M_2(x) \leftrightarrow \bigcirc(M_1(x) \wedge \mathrm{Right}(x \ominus 1)))$
$\forall x(M_3(x) \leftrightarrow M_1(x) \wedge \neg\mathrm{Right}(x \ominus 1))$
$\Box\forall x(\bigcirc M_3(x) \leftrightarrow \bigcirc(M_1(x) \wedge \neg\mathrm{Right}(x \ominus 1)) \vee M_4(x))$
$\forall x(M_4(x) \leftrightarrow M_3(x) \wedge \neg(\mathrm{Reclaim}(x \ominus 1) \vee \mathrm{Enough}(x)))$
$\Box\forall x(\bigcirc M_4(x) \leftrightarrow \bigcirc(M_3(x) \wedge \neg(\mathrm{Reclaim}(x \ominus 1) \vee \mathrm{Enough}(x))))$

Figure 7.2: Transition scheme and axioms for clause 7.10

(1)	$\neg\text{Eat}(x \ominus 1) \rightarrow \neg\text{Right}(x \ominus 1)$	$(I1)$
(2)	$\text{Eat}(x \ominus 1) \rightarrow \Diamond\text{Enough}(x \ominus 1)$	(7.11)
(3)	$\text{Enough}(x \ominus 1) \rightarrow \neg\text{Right}(x \ominus 1)$	$(I2)$
(4)	$\text{Waiting}(x) \rightarrow \Diamond\neg\text{Right}(x \ominus 1)$	(1), (2), (3)
(5)	$\text{Waiting}(x) \rightarrow \text{Waiting}(x)$ **unless** $\text{Eat}(x) \vee \text{Enough}(x)$	$(I3)$
(6)	$\text{Waiting}(x) \rightarrow \Diamond((\text{Waiting}(x) \wedge \neg\text{Right}(x \ominus 1)) \vee \text{Eat}(x) \vee \text{Enough}(x))$	(4), (5)
(7)	$\text{Waiting}(x) \wedge \neg\text{Right}(x \ominus 1) \rightarrow \text{Left(x)}$	(7.10)
(8)	$\text{Waiting}(x) \wedge x \ominus \mathit{pref} \leq k \rightarrow \Diamond((\text{Waiting}(x) \wedge \text{Left(x)} \wedge \text{x} \ominus \mathit{pref} \leq \text{k}) \vee \text{Eat}(x) \vee \text{Enough}(x))$	(6), (7), $(I4)$

What has been shown is that every waiting philosopher, if not already satisfying the assertion $(*)$ above, at least will eventually receive his left fork without falling in the "preference hierarchy" of the butler. Can it happen that he retains the left fork forever (without ever getting the right fork)?

(9)	$\Box\text{Left(x)} \wedge \Diamond\neg\text{Left}(\text{x} \oplus 1) \rightarrow \Diamond(\text{Eat(x)} \vee \text{Enough(x)})$	(7.7)
(10)	$\text{Eat}(x) \rightarrow \Diamond\text{Enough}(x)$	(7.11)
(11)	$\Box\text{Left(x)} \rightarrow \Box\text{Left}(\text{x} \oplus 1) \vee \Diamond(\text{Enough}(x) \wedge \neg\text{Hungry}(x))$	(9), (10), (7.12)
(12)	$\Box\text{Left(x)} \rightarrow \Box\text{Left}(\text{x} \oplus 1) \vee \Diamond\neg\text{Left(x)}$	(11), $(I5)$
(13)	$\Box\text{Left(x)} \rightarrow \Box\text{Left}(\text{x} \oplus 1)$	(12)
(14)	$\forall x(\Box\text{Left(x)} \rightarrow \Box\text{Left}(\text{x} \oplus 1))$	(13)
(15)	$\forall x(\Box\text{Left(x)} \rightarrow \Box\text{Left}(\text{x} \oplus 1)) \rightarrow \forall x(\Box\text{Left(x)} \rightarrow \forall \text{y}\Box\text{Left(y)})$	(data)
(16)	$\Box\text{Left(x)} \rightarrow \forall \text{y}\Box\text{Left(y)}$	(14), (15)
(17)	$\Box\text{Left(x)} \rightarrow \text{Left}(\mathit{pref}) \wedge \text{Left}(\mathit{pref} \oplus 1) \wedge \text{Left}(\mathit{pref} \oplus 2)$	(16)
(18)	$\Box\text{Left(x)} \rightarrow \text{Left}(\mathit{pref}) \wedge \text{Left}(\mathit{pref} \oplus 1) \wedge \neg\text{Right}(\mathit{pref} \oplus 1)$	(17), $(I6)$
(19)	$\Box\text{Left(x)} \rightarrow \exists\text{yReclaim(y)}$	(pred)
(20)	$\text{Reclaim}(y) \rightarrow \circ(\text{Right}(y) \vee \text{Enough}(y))$	(7.9)
(21)	$\exists y\text{Reclaim}(y) \rightarrow \circ\exists y\neg\text{Left(y)}$	(20), $(I6)$, (7.12), $(I5)$
(22)	$\Diamond\neg\text{Left(x)}$	(16), (19), (21)

No, every philosopher eventually loses his left fork. But according to the invariant (I7), there are only two possible reasons for losing a left fork, both of which establish assertion $(*)$:

(23)	$\text{Left(x)} \rightarrow \Diamond(\text{Reclaim}(\text{x} \ominus 1) \vee \text{Enough(x)})$	$(22), (I7)$
(24)	$\text{Reclaim}(x \ominus 1) \rightarrow x \ominus \mathit{pref} = 1 \wedge \circ(x \ominus \mathit{pref} = 0)$	$(7.9), (\text{data})$
(25)	$\text{Reclaim}(x \ominus 1) \wedge x \ominus \mathit{pref} \leq k \rightarrow \circ(x \ominus \mathit{pref} < k)$	$(24), (\text{data})$
(26)	$\text{Waiting}(x) \wedge \text{Left(x)} \wedge \text{x} \ominus \mathit{pref} \leq \text{k} \rightarrow$ $\Diamond((\text{Waiting}(x) \wedge x \ominus \mathit{pref} < k) \vee \text{Eat}(x) \vee \text{Enough}(x))$	$(23), (25), (I4)$
(27)	$\text{Waiting}(x) \wedge x \ominus \mathit{pref} \leq k \rightarrow$ $\Diamond((\text{Waiting}(x) \wedge x \ominus \mathit{pref} < k) \vee \text{Eat}(x) \vee \text{Enough}(x))$	$(8), (26)$

7.2 Filtering two streams

Our second example illustrates the composition rule of section 6.4 by combining two modules whose correctness mutually depends on each other. Consider a simple filter program receiving two streams of integer values (modelled by the environment predicates In_1 and In_2, respectively) which have to be assigned to two output streams (modelled by the system-owned predicates Out_1 and Out_2) such that Out_1 outputs the negative values received by either In_1 or In_2, and Out_2 similarly outputs the positive values. Every nonzero value received by the system eventually has to be output by either Out_1 or Out_2 (we do not care about the multiplicity of values). Obviously, for this requirement to be implementable the environment has to make sure that a value input by either In_1 or In_2 is observed by the system. A way to achieve this is to feed every value to the system until it appears at one of the outputs. Formally, the specification for the filter program is

(7.14) $\forall x \Box(Out_1(x) \rightarrow x < 0)$

(7.15) $\forall x \Box(Out_2(x) \rightarrow x > 0)$

(7.16) $\forall x \Box((In_1(x) \vee In_2(x)) \wedge x \neq 0 \rightarrow \Diamond(Out_1(x) \vee Out_2(x))),$

assuming

(7.17) $\forall x \Box(In_1(x) \rightarrow In_1(x) \text{ unless } (Out_1(x) \vee Out_2(x)))$ and

(7.18) $\forall x \Box(In_2(x) \rightarrow In_2(x) \text{ unless } (Out_1(x) \vee Out_2(x))).$

The problem is solved by dividing the system into two modules as shown in figure 7.3.

Module M_1 receives the input stream In_1 and outputs the negative values by Out_1 while M_2 receives In_2 and owns the output channel Out_2 for positive values. The modules communicate the values which have to be output by the other module using the channels $Send_1$ and $Send_2$. To make sure that every value sent has been observed by the other module, it is sent until it has been echoed via the acknowledgement channel by the partner module. The νTL

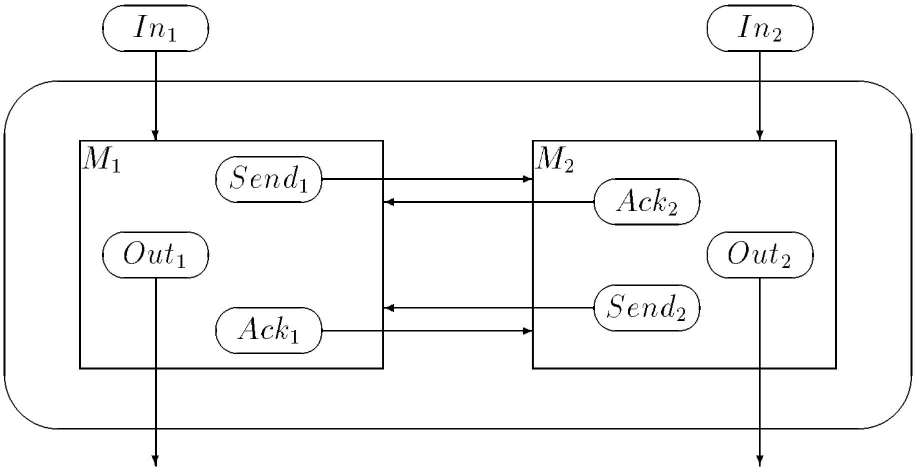


Figure 7.3: Structure of the filter program

programs implementing M_1 and M_2 are given in table 7.2. Observe that in this example we are using the standard predicate $<$ on integers in the way indicated at the end of chapter 5. To follow the strict syntax definition 5.3 for program clauses, the predicate $x < y$ would have to be realized by introducing a function $lt : \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$ defined by $lt(x, y) = 1$ iff $x < y$, and replacing $x < y$ by $lt(x, y) = 1$. Using the standard predicate $<$ improves the readability of the formulas. Otherwise, the formulas constituting Φ_1 and Φ_2 are acceptable program clauses which do not induce any conditions on the dependency order.

Both νTL programs implementing the modules are trivially $\mathcal{U}$-compatible (for any environment assumption $\mathcal{U}$) because no state components are modelled by flexible variables.

Let us now demonstrate part of the correctness of $M_1 \parallel M_2$. Specifically, we prove that all initialized runs of $M_1 \parallel M_2$ satisfy (7.14) and (7.15) by applying the composition rule of section 6.4, choosing F_1 and F_2 as

$$\begin{aligned} F_1 &\equiv \Box\forall x\,(Out_1(x) \to x < 0) \wedge \Box\forall x\,(Send_1(x) \to x > 0) \quad \text{and} \\ F_2 &\equiv \Box\forall x\,(Out_2(x) \to x > 0) \wedge \Box\forall x\,(Send_2(x) \to x < 0). \end{aligned}$$

For the environment assumptions Ψ_1 and Ψ_2, we take

$$\begin{aligned} \Psi_1 &\equiv \Box\forall x\,(Send_2(x) \to x < 0) \quad \text{and} \\ \Psi_2 &\equiv \Box\forall x\,(Send_1(x) \to x > 0), \end{aligned}$$

and the initialization formulas are defined by

$$\begin{aligned} \mathcal{I}_1 &\equiv (\forall x \neg Send_1(x) \wedge \forall x \neg Out_1(x)) \;\mathsf{unless}\; sch_1 \quad \text{and} \\ \mathcal{I}_2 &\equiv (\forall x \neg Send_2(x) \wedge \forall x \neg Out_2(x)) \;\mathsf{unless}\; sch_2. \end{aligned}$$

Module M_1 (program Φ_1):	
$(M11)$	$\Box(In_1(x) \rightarrow (x > 0 \rightarrow Send_1(x) \text{ unless } Ack_2(x)))$
$(M12)$	$\Box(In_1(x) \rightarrow (x < 0 \rightarrow Out_1(x)))$
$(M13)$	$\Box(Send_2(x) \rightarrow Ack_1(x) \wedge Out_1(x))$
Module M_2 (program Φ_2):	
$(M21)$	$\Box(In_2(x) \rightarrow (x < 0 \rightarrow Send_2(x) \text{ unless } Ack_1(x)))$
$(M22)$	$\Box(In_2(x) \rightarrow (x > 0 \rightarrow Out_2(x)))$
$(M23)$	$\Box(Send_1(x) \rightarrow Ack_2(x) \wedge Out_2(x))$

Table 7.2: νTL implementations of the filter modules

Observe that $\mathcal{I}_1$ and $\mathcal{I}_2$ fit the general scheme Θ_i unless sch_i for initialization formulas identified in chapter 6 because for $i = 1, 2$, Θ_i is equivalent to $\forall x(Send_i(x) \leftrightarrow \mathsf{f}) \wedge \forall x(Out_i(x) \leftrightarrow \mathsf{f}$, and f corresponds to the empty disjunction.

Clearly, Ψ_1 and Ψ_2 are satisfiable. To apply the composition rule (for the system environment assumption $\Psi \equiv \mathsf{t}$), proving that every initialized run of $M_1 \parallel M_2$ satisfies $\tau_1(F_1) \wedge \tau_2(F_2)$, we have to show that

$$\Psi_1, \Phi_1^c \models F_1 \quad \text{and} \quad \Psi_2, \Phi_2^c \models F_2$$

as well as

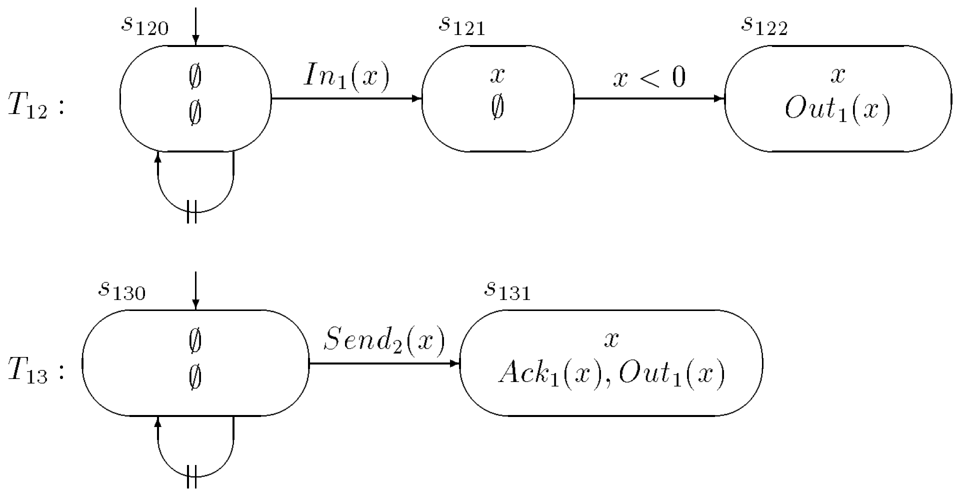
$$\begin{array}{l} (P0), (P1), (P2), \mathcal{I}_1, \tau_1(F_1) \models \tau_2(\Psi_2) \quad \text{and} \\ (P0), (P1), (P2), \mathcal{I}_2, \tau_2(F_2) \models \tau_1(\Psi_1). \end{array}$$

We will explicitly carry out the proofs for $\Psi_1, \Phi_1^c \models \Box\forall x\,(Out_1(x) \rightarrow x < 0)$ and for $(P0), (P1), (P2), \mathcal{I}_2, \tau_2(F_2) \models \tau_1(\Psi_1)$, the remaining proofs being analogous.

First, we need some completion axioms for program Φ_1. Consider the transition schemes T_{12} and T_{13} associated with the clauses $(M12)$ and $(M13)$ which, together with the corresponding transition axioms according to definition 5.26, are shown in figure 7.4.

Observing that Out_1 does not appear in the program clause $(M11)$, hence that $Def_{11}(Out_1) = \emptyset$ (cf. the notation of definition 5.29), the definition axiom for Out_1 thus becomes

$$\begin{array}{rcl} (\mathrm{DEF}_{Out_1}) & \equiv & \Box\forall y(Out_1(y) \leftrightarrow \exists x(M_{122}(x) \wedge y = x) \vee \exists x(M_{131}(x) \wedge y = x)) \\ & \equiv & \Box\forall x(Out_1(x) \leftrightarrow M_{122}(x) \vee M_{131}(x)). \end{array}$$



Transition axioms for T_{12}:	
M_{120}	$\Box(\bigcirc M_{120} \leftrightarrow M_{120})$
$\forall x(M_{121}(x) \leftrightarrow M_{120} \wedge In_1(x))$	$\Box\forall x(\bigcirc M_{121}(x) \leftrightarrow \bigcirc(M_{120} \wedge In_1(x)))$
$\forall x(M_{122}(x) \leftrightarrow M_{121}(x) \wedge x < 0)$	$\Box\forall x(\bigcirc M_{122}(x) \leftrightarrow \bigcirc(M_{121}(x) \wedge x < 0))$
Transition axioms for T_{13}:	
M_{130}	$\Box(\bigcirc M_{130} \leftrightarrow M_{130})$
$\forall x(M_{131}(x) \leftrightarrow M_{130} \wedge Send_2(x))$	$\Box\forall x(\bigcirc M_{131}(x) \leftrightarrow \bigcirc(M_{130} \wedge Send_2(x)))$

Figure 7.4: Transition schemes and axioms for $(M12)$ and $(M13)$

It is easy to see, using the transition axioms and (DEF_{Out_1}), that

$$\Phi_1^c \models \Box\forall x(Out_1(x) \to x < 0 \vee Send_2(x)),$$

which together with the environment assumption Ψ_1 yields

$$\Phi_1^c, \Psi_1 \models \Box\forall x(Out_1(x) \to x < 0),$$

establishing the first proof.

For the proof of $(P0), (P1), (P2), \mathcal{I}_2, \tau_2(F_2) \models \tau_1(\Psi_1)$, recall from example 6.6 that

$$(P1), (P2), \tau_2(\Box F) \models \Box F \ \mathsf{atnext}\ sch_2$$

if F is a state formula all of whose flexible symbols are among the state components owned by module M_2, a condition met by the formula F_2. Hence, it follows that

$$(P1), (P2), \tau_2(F_2) \models (\Box\forall x\,(Send_2(x) \to x < 0))\ \mathsf{atnext}\ sch_2.$$

On the other hand, the initialization formula $\mathcal{I}_2$ implies $\forall x\,(Send_2(x) \to x < 0)\ \mathsf{unless}\ sch_2$, hence we conclude

$$(P1), (P2), \mathcal{I}_2, \tau_2(F_2) \models \Box\forall x\,(Send_2(x) \to x < 0)$$

which in turn can easily be shown to imply

$$(P1), (P2), \mathcal{I}_2, \tau_2(F_2) \models \tau_1(\Psi_1).$$

Having carried out the remaining proofs required for the applicability of the composition rule, one may conclude that all initialized runs of $M_1 \parallel M_2$ obeying the initialization formulas $\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy $\tau_1(F_1) \wedge \tau_2(F_2)$ which, again by example 6.6, may be strenghtened to

$$(\Box\forall x\,(Out_1(x) \to x < 0))\ \mathsf{atnext}\ sch_1 \wedge (\Box\forall x\,(Out_2(x) \to x > 0))\ \mathsf{atnext}\ sch_2.$$

Taking into account the initialization formulas, one finally arrives at (7.14) and (7.15).

For the proof of (7.16), weak fairness assumptions for both modules have to be introduced, ensuring that no module will eventually be completely locked out, and the environment assumptions (7.17) and (7.18) have to be used.

Chapter 8

Conclusion

Let us conclude by discussing the obtained results and comparing them with those of other authors. The central result has been the definition of a class of temporal logic formulas which can be assigned an imperative meaning concrete enough to form the base of an interpreter. In order to understand the restrictions imposed on these "program clauses", the connection between the syntactical form of a formula and the type of the property expressed by it had to be investigated. It turned out that the main requirement for a formula to be a program clause was that it expressed a deterministic safety formula. (Strictly, this has only been shown for propositional program clauses — first-order counterparts to Streett automata do not appear in the literature.) The introduction of the dependency order on the state components and its use in the definitions of proper program clauses (definitions 4.2 and 5.3) represents an additional technical requirement which serves to rule out cyclic dependencies between the values of state components in one single state. In the propositional case, this mainly allows the interpreter to operate somewhat more efficiently (arbitrary assignments of truth values to "undefined" proposition letters generate models). In the first-order case, the restriction is crucial to ensure finite predicate extensions in every state (otherwise, a clause like $P(x) \rightarrow P(f(x))$ could produce an infinite extension, and the computation of one single state could not be completed in finite time). One may wonder if the restrictions on program clauses could be loosened significantly.

For example, the METATEM language [11] which is also based on the idea of constructing models for temporal logic specifications allows, in effect, arbitrary temporal logic formulas as input to the interpreter, the restrictions imposed concerning only the syntax without decreasing the expressiveness. The price to be paid, as the authors themselves acknowledge, is the possibility of unbounded backtracking, even for propositional programs. The well-known small-model property of temporal logic does not help here because the model is not determined by the interpreter alone, but results from an interaction of the environment and the system. Consequently, the interpreter has to heuristically choose how to complete the construction of the current state without knowing which choices may lead to a successful model construction. In the first-order case, the situation becomes even worse since in general infinitely many possibilities to choose from exist at every state. As a more technical drawback, the METATEM interpreter requires the complete computation history up to some state to be examined to determine which clauses should fire in that state. In contrast, the interpreter algorithms 4.32 and 5.32 only need to consult the current marking, the relevant part of the history being coded into the transition graphs (or schemes). On the other hand, METATEM programs tend to be more concise. The emphasis of the present approach has been to allow the developed program to be interpreted as efficiently as possible while maintaining the goal to offer higher-level language constructs, abstracting away from explicit state transitions that have to be described by usual programming notations.

It has been motivated in the discussion of section 4.1 under the heading "absence of essential non-determinism" why disjunctions have been banned from appearing in pre-clause bodies. If disjunctions were to be allowed while still aiming to avoid observable backtracking, it would have to be ensured that no choice of disjuncts precluded the generation of a model for the given program. Therefore, the notion of compatibility (cf. the definitions 4.31 and 5.31) would have to be strengthened: It would no longer suffice to construct conflict graphs (or schemes) for pairs of program clauses, but arbitrary combinations of program clauses would have to be checked. The language designer has to decide on whether this is a price worth paying for the corresponding gain in expressivity.

Other prominent representatives of the "executable temporal logic" paradigm include MOSZKOWSKI'S [41] programming language Tempura and TANG'S [52] systems XYZ/E and XYZ/D. These languages are essentially based on formulas of the pure $\circ$-fragment, possibly with an outermost $\Box$-operator. Tempura, defined over an interval-based instead of a state-based semantics, moreover offers an iteration operator to express while-loops. Still, these languages thus appear as strictly less expressive sublanguages of the class of executable formulas defined here.

More in line with classical logic programming languages is the TEMPLOG language presented in ABADI [1] and further studied in BAUDINET [12] whose interpreter uses a resolution proof system for a sublanguage of the $(\circ, \Box)$-fragment of temporal logic. The admissible TEMPLOG program clauses resemble of (nested) Horn clauses. Queries may only contain $\circ$- and $\Diamond$-operators, and the occurrence of $\Box$-operators in bodies is extremely restricted. In fact, these syntactic restrictions avoid an incompleteness proof similar to the one given in chapter 3, resulting in a complete logic. However, the intended applications appear to be somewhat different from our approach in that queries are posed which can be answered over finite prefixes of runs, hence computations do not generate potentially infinite runs of the system described by the program; moreover, no environment interaction is assumed.

Besides their motivation for understanding the restrictions on executable program clauses, the more theoretical results of this thesis may be interesting in their own right. Specifically, the novel characterization of safety properties given in section 2.6 provides a quite natural and easily checked condition for determining whether a given formula expresses a safety property (and will therefore be proven by some invariance argument in contrast to well-founded induction or proof chains for liveness properties). Another characterization of safety properties in temporal logic which has been known for some time (see, e.g., MANNA & PNUELI [37]) uses past-time operators and therefore sometimes requires a somewhat unnatural reformulation of the given formula. For example, the formula p unless $q \equiv \nu X.\neg q \rightarrow p \wedge \circ X$ (p and q being proposition letters) can immediately be classified as expressing a safety property by theorem 2.30, whereas it has to be shown to be equivalent to $\Box(\neg p \rightarrow \diamondminus q)$ to apply the criterion of MANNA & PNUELI [37] where $\diamondminus$ is the past time counterpart to the operator $\Diamond$, i.e. $\diamondminus F$ is true at some state s_i if F is true at some state s_j for $j \leq i$. A natural question to study is to try and give similar characterizations of the other classes of the property hierarchy studied in MANNA & PNUELI [37] in terms of alternations of fixpoint operators in νTL formulas.

A prominent topic of research in the field of formal development methods is the compositionality of development formalisms, inspired by the clear and syntax-directed proof rules of, e.g., the Hoare calculus (HOARE [23]). In contrast with such elegance, the rule to derive properties of systems composed of several modules given in lemma 6.5 fails miserably. It should however be noted that for certain classes of properties (specifically, for stuttering-closed properties) much better rules can be found — where the parallel composition of modules corresponds to the conjunction of single modules' properties to yield properties of the entire system (see, e.g., MANNA & PNUELI [38] or ABADI & LAMPORT [3]). Also, example 6.6 points in this direction. It would be interesting to find a syntactic characterization of stuttering-closed properties in νTL analogous to the syntactic characterization of safety properties given in section 2.6.

Another problem associated with the specification and development of (not only reactive) programs using logical formalisms which has traditionally received much attention is the frame problem: How can it be guaranteed that an action affects only the relevant part of the state? This problem is partly solved by the expressive means of temporal logic which allows to talk about states and state changes, specifically offering appropriate operators for stating for how long a condition should stay true. For example, the clauses (7.7) and

(7.10) of the dining-philosophers example of chapter 7 explicitly state (necessary) conditions for philosophers to lose their forks, requiring them to keep the forks at least until these conditions become true. In the first-order case, this is not enough because the predicate extensions additionally have to be bounded from above. For this reason, attention has been restricted to normal models (definition 5.25).

Certainly, much remains to be done before the development framework outlined here becomes acceptable for the development of "real-world" programs. In particular, machine assistance is absolutely necessary to carry out the proofs required during the development process. Already the toy examples of chapter 7 were too big to present a complete verification. On the other hand, much can be done automatically such as generating the formulas associated with conflict graphs (schemes) which have to be proven to ensure the compatibility of νTL programs, or generating the completion axioms of definition 5.29. Also, despite the inherent incompleteness of first-order temporal logic, the proofs actually required for program verification are usually not very complex, and it should be possible to develop automatic or semi-automatic provers to assist the programmer. A promising starting point for such a temporal logic prover has been developed in Feess [16].

The specification of data types and their associated operations has been completely ignored. They were simply assumed to be provided by the underlying first-order structure, and appropriate "data" axioms were used in the verification of properties. Of course, any full-fledged programming language has to provide at least a basic assortment of data types and structuring mechanisms to construct new, problem-oriented data types. In the realm of reactive systems, additional data types such as channels, signals and messages occur naturally and should be integrated into the language, including specific axioms in the proof rules used for the verification of programs.

Another issue to deal with concerns the acceptance of temporal logic formulas by system developers. In particular, the bare νTL syntax and semantics quickly becomes unwieldy for larger formulas. The reader will have noticed that the examples have been presented using only derived temporal operators whose semantics can be understood quite independently from fixpoint definitions. On the other hand, the flexibility of the νTL language allows to define virtually every reasonable temporal operator, and its uniformity makes it an ideal candidate for mechanical manipulation by computers.

In any case, I hope that this thesis proves interesting enough to pose new questions for further study.

Bibliography

[1] M. Abadi: *Temporal-Logic Theorem Proving*. PhD thesis, Department of Computer Science, Stanford University (1987)

[2] M. Abadi: *The Power of Temporal Proofs*. DEC SRC Research Report **30** (1988). Revised Version in: TCS **65**, pp. 35–84 (1989)

[3] M. Abadi, L. Lamport: *Composing Specifications*; in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): Stepwise Refinement of Distributed Systems — Models, Formalisms, Correctness. LNCS 430, pp. 1–41, Springer-Verlag (1990)

[4] M. Abadi, L. Lamport, P. Wolper: *Realizable and unrealizable program specifications*; in: Proc. International Conference on Automata, Languages and Programming (G. Ausiello et al., eds.). LNCS 372, pp. 1–17, Springer-Verlag (1989)

[5] M. Abadi, Z. Manna: *Nonclausal Deduction in First-Order Temporal Logic*. Journal of the ACM **37**, No. 2, pp. 279–317 (1990)

[6] W. Ackermann: *Solvable Cases of the Decision Problem*. Amsterdam: North Holland (1954)

[7] B. Alpern, F.B. Schneider: *Recognizing safety and liveness*. Distributed Computing **2**, pp. 117–126 (1987)

[8] B. Alpern, F.B. Schneider: *Verifying temporal properties without temporal logic*. ACM Transactions on Programming Languages and Systems **11**, pp. 147–167 (1989)

[9] R. Alur, T.A. Henzinger: *A really temporal logic*; in: Proc. of the 30^{th} IEEE Symposium on Foundations of Computer Science, pp. 164–169 (1989)

[10] B. Banieqbal, H. Barringer: *Temporal Logic with Fixed Points*; in: Proc. Temporal Logic in Specification (B. Banieqbal et al., ed.). LNCS 398, pp. 62–74, Springer-Verlag (1989)

[11] H. Barringer, M. Fisher, D. Gabbay, G. Gough, R. Owens: MetateM*: A Framework for Programming in Temporal Logic*; in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): Stepwise Refinement of Distributed Systems — Models, Formalisms, Correctness. LNCS 430, Springer-Verlag (1990)

[12] M. Baudinet: *Temporal Logic Programming is Complete and Expressive*; in: Proc. ACM Conf. on Princinples of Programming Languages, pp. 267–280 (1989)

[13] J. Bradfield, C. Stirling: *Verifying Temporal Properties of Processes*; in: CONCUR '90 — Theories of Concurrency: Unification and Extension. LNCS 458, pp. 115–125, Springer-Verlag (1990)

[14] K.M. Chandy, J. Misra: *Parallel Program Design: A Foundation*. Reading, Mass.: Addison-Wesley Publishing Company (1988)

[15] C.J. Date: *An Introduction to Database Systems, Vol. I*. Reading, Mass.: Addison-Wesley Publishing Company (Systems Programming Series) (^{3}1985)

[16] T. Feess: *Ein Tableauverfahren für die temporale Hornklausel-Prädikatenlogik*. Diplomarbeit, TU München (1991)

[17] D.M. Gabbay: *Modal and Temporal Logic Programming*; in: Temporal Logics and their Applications (A. Galton, ed.). London: Academic Press (1987)

[18] D.M. Gabbay, A. Pnueli, S. Shelah, J. Stavi: *On the temporal analysis of fairness*; in: Proc. 7^{th} ACM Symposium on Principles of Programming Languages, pp. 163–173 (1980)

[19] H. Gallaire, J. Minker, J.-M. Nicolas: *Logic and Databases: A Deductive Approach*. Computing Surveys **16**, No. 2, pp. 152–185 (1984)

[20] J.W. Garson: *Quantification in modal logic*; in: Handbook of Philosophical Logic, vol. II. Dordrecht: Reidel, pp. 249–308 (1984)

[21] D. Harel: *Effective Transformations on Infinite Trees, with Applications to High Undecidability*. Journal of the ACM **33**, No. 1, pp. 224–248 (1986)

[22] D. Harel, A. Pnueli, J. Stavi: *Propositional Dynamic Logic of Nonregular Programs*. Journal of Computer and System Sciences **26** (1983), pp. 222–243

[23] C. A. R. Hoare: *An axiomatic basis for computer programming.* Communications of the ACM **12**, pp. 576–580 (1969)

[24] S. Katz, D. Peled: *Interleaving Set Temporal Logic.* Theoretical Computer Science **75**, pp. 263–287 (1990)

[25] R. A. Kowalski: *Predicate Logic as a Programming Language*; in: Proc. of the IFIP Congress, pp. 569–574 (1974)

[26] D. Kozen: *Results on the propositional μ-calculus*; in: E. Schmidt (ed.), Proc. 9^{th} International Colloqium on Automata, Languages, and Programming 1982, LNCS 140, pp. 348–352, Springer-Verlag. Revised Version in: Theoretical Computer Science **27**, pp. 333–354 (1983)

[27] D. Kozen, R. Parikh: *A decision procedure for the propositional μ-calculus*; in: Proc. Workshop on Logics of Programs (E. Clarke, D. Kozen, eds.). LNCS 164, pp. 313–325, Springer-Verlag (1984)

[28] F. Kröger: *A uniform logical basis for the description, specification and verification of programs*; in: Formal Descriptions of Programming Concepts, E. J. Neuhold (ed.). Amsterdam: North Holland Publishing Company (1978)

[29] F. Kröger: *Temporal Logic of Programs.* EATCS Monographs on Theoretical Computer Science **8**. Berlin: Springer-Verlag (1987)

[30] F. Kröger: *On the Interpretability of Arithmetic in Temporal Logic.* TCS **73**, pp. 47–60 (1990)

[31] L. Lamport: *Proving the correctness of multiprocess programs.* IEEE Transactions on Software Engineering SE-3,2, pp. 125–143 (1977)

[32] L. H. Landweber: *Decision problems for ω-automata.* Mathematical System Theory **4**, pp. 376–384 (1969)

[33] K. G. Larsen: *Proof Systems for Hennessy-Milner Logic with Recursion*; in: Proc. Coll. on Trees in Algebra and Programming (M. Dauchet, M. Nivat, eds.). LNCS **299**, pp. 215–230, Springer-Verlag (1988)

[34] O. Lichtenstein, A. Pnueli, L. Zuck: *The Glory of the Past*; in: Logics of Programs. LNCS **193**, pp. 196–218, Springer-Verlag (1985)

[35] J. W. Lloyd: *Foundations of Logic Programming.* Berlin: Springer-Verlag (^{2}1987)

[36] Z. Manna, A. Pnueli: *Completing the temporal picture*; in: Proc. of the 16^{th} International Colloqium on Automata, Languages and Programming (G. Ausiello et al., eds.). LNCS 372, pp. 534–558, Springer-Verlag (1989)

[37] Z. Manna, A. Pnueli: *A Hierarchy of Temporal Properties*; in: Proc. ACM Symposium on Principles of Distributed Computing (1990)

[38] Z. Manna, A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems — Specification.* New York etc.: Springer-Verlag (1992)

[39] Z. Manna, P. Wolper: *Synthesis of Communicating Processes from Temporal Logic Specifications.* ACM Transactions on Programming Languages **6**, pp. 68–93 (1984)

[40] S. Merz: *Decidability and Incompleteness Results for First-Order Temporal Logics of Linear Time.* To appear in: Journal of Applied Non-Classical Logics (1992)

[41] B. Moszkowski: *Executing Temporal Logic Programs.* Cambridge (UK): Cambridge University Press (1986)

[42] B. Paech: *Extending temporal logic by explicit concurrency*; in: Proc. Mathematical Foundations of Computer Science. LNCS 520, pp. 377–386, Springer-Verlag (1991)

[43] A. Pnueli: *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends*; in: Current Trends in Concurrency (J.W. de Bakker et al., eds.). LNCS **224**, pp. 510–585, Springer-Verlag (1986)

[44] A. Pnueli, R. Rosner: *On the Synthesis of a Reactive Module*; in: Proc. International Conference on Automata, Languages and Programming (G. Ausiello et al., eds.). LNCS 372, pp. 1–17, Springer-Verlag (1989)

[45] V.R. Pratt: *A decidable μ-calculus*; in: Proc. 22^{nd} IEEE Symposium on Foundations of Computer Science, pp. 421–427 (1981)

[46] R. Reiter: *On Closed-World Data Bases*; in: Logic and Data Bases (H. Gallaire and J. Minker, eds.). New York: Plenum Press, pp. 55–76 (1978)

[47] C. Stirling: *Modal and Temporal Logics.* Report ECS-LFCS-91-157, Laboratory for Foundations of Computer Science, Dept. of Computer Science, Univ. of Edinburgh (1991)

[48] A. Szałas: *Concerning the Semantic Consequence Relation in First-Order Temporal Logic*. TCS **47**, pp. 329–334 (1986)

[49] A. Szałas: *A Complete Axiomatic Characterization of First-Order Temporal Logic of Linear Time*. TCS **54**, pp. 199–214 (1987)

[50] A. Szałas: *Axiomatizing fixpoint logics*. Information Processing Letters **41**, pp. 175–180 (1992)

[51] A. Szałas, L. Holenderski: *Incompleteness of First-Order Temporal Logic with Until*. TCS **57**, pp. 317–325 (1988)

[52] Z. Tang: *Toward a Unified Logic Basis for Programming Languages*; in: IFIP Congress 83 (R. E. A. Mason, ed.). Amsterdam: Elsevier Publishers B.V. (1983)

[53] A. Tarski: *A lattice-theoretical fixpoint theorem and its applications*. Pacific Journal of Mathematics **5**, pp. 285–309 (1955)

[54] W. Thomas: *Automata on Infinite Objects*; in: Handbook of Theoretical Computer Science, pp. 133–191. Amsterdam: Elsevier Science Publishers B.V. (1990)

[55] M. Vardi: *A Temporal Fixpoint Calculus*; in: Proc. of the 15^{th} ACM Symposium on the Principles of Programming Languages, pp. 250–260 (San Diego, California, 1988)

[56] M. Vardi: *Verification of concurrent programs: the automata-theoretic framework*. Annals of Pure and Applied Logic **51**, pp. 79–98 (1991)

[57] K. Wagner: *On ω-regular sets*. Information and Control **43**, pp. 123–177 (1979)

[58] P. Wolper: *Synthesis of Communicating Processes from Temporal Logic Specifications*. Ph.D. thesis, Stanford University (1982)

[59] P. Wolper: *Temporal Logic Can Be More Expressive*. Information and Control **56**, pp. 72–99 (1983)

Index