COMPOSITIONAL VERIFICATION OF REACTIVE AND REAL-TIME SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Edward Chang
July 1995

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Zohar Manna
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Amir Pnueli

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Yoav Shoham

Approved for the University Committee on Graduate Studies:

Abstract

This thesis presents a compositional methodology for the verification of reactive and real-time systems. The correctness of a given system is established from the correctness of the system's components, each of which may be treated as a system itself and further reduced. When no further reduction is possible or desirable, global techniques for verification may be used to verify the bottom-level components.

Transition modules are introduced as a suitable compositional model of computation. Various composition operations are defined on transition modules, including parallel composition, sequential composition, and iteration. A restricted assumption-guarantee style of specification is advocated, wherein the environment assumption is stated as a restriction on the environment's next-state relation. Compositional proof rules are provided in accordance with the safety-progress hierarchy of temporal properties.

The compositional framework is then extended naturally to real-time transition modules and discrete-time metric temporal logic.

Acknowledgements

I would like to thank Prof. Zohar Manna, my advisor for the past four years, for the invaluable wisdom and guidance he has provided me. His dedication and unwavering enthusiasm for research has been a continuous source of inspiration. I am also especially grateful for the pleasant and extremely productive visit to the Weizmann Institute that he made possible. I am certain that those months in Israel will always remain a highlight of my graduate school years.

I would also like to thank Prof. Amir Pnueli, who has been so generous with his insight these past few years. He has contributed immeasurably by helping to solve all the difficult problems and by suggesting new directions of research. I consider myself extremely fortunate to have had the opportunity to work with him. I have also had several enlightening discussions with Prof. Tom Henzinger of Cornell, and I would like to thank him for his help. Finally, it has been a pleasure working with the recently-evolved REACT group here at Stanford, and I hope that we will continue the swift progress that we have made together.

My undergraduate advisor at Brown, Prof. Andries van Dam, deserves all the credit for getting me started in computer science research. Working with the Graphics Group was both immensely fun and wonderfully challenging.

Finally, I would like to thank my friends and my family, especially my parents, for their constant support and encouragement. I have called on them many times during my years at Stanford, and they have always been there.

Contents

Abstract **iv**

Acknowledgements **v**

1 Introduction **1**

2 The Global Framework **3**

2.1 Fair Transition Systems 3

2.1.1 The Idling Transition 4

2.1.2 Computations of a Fair Transition System 5

2.2 A Simple Programming Language 5

2.2.1 Statement Labels 7

2.2.2 Semantics of the Programming Language 8

2.2.3 Transitions of a Statement 8

2.3 Temporal Logic 10

2.3.1 Classification of Formulas 12

2.4 Verification 17

2.4.1 Verification Conditions 17

2.4.2 Proof Rules for Safety Properties 17

2.4.3 Proof Rules for Response Properties 18

2.4.4 Proof Rules for Reactivity Properties 18

3 Parallel Composition **19**

3.1 Transition Modules 19

3.1.1 The Environment Transition 21
3.1.2 Computations of a Transition Module 21
3.1.3 Parallel Composition 22
3.2 The Environment Restriction 23
3.3 Verification 24
3.3.1 Proof Rules for Safety Properties 25
3.3.2 Proof Rules for Response Properties 33
3.3.3 Layered Decomposition 36
3.3.4 Proof Rules for Reactivity Properties 38
3.3.5 Embedded Transition Modules 38
3.4 Asynchronous Communication 39
3.4.1 The Compassion Requirement 40
3.4.2 Compassionate Proof Rules 41
3.5 Synchronous Communication 43
3.5.1 Parallel Composition 44
3.5.2 Verification 45
3.6 Related Work 47

4 Sequential Composition and Iteration **49**
4.1 Transition Modules 50
4.1.1 The Environment Transition 52
4.1.2 Composing Transition Modules 52
4.2 Anchored Composition 55
4.2.1 Computations of a Transition Module 55
4.2.2 Temporal Logic with Follows 57
4.2.3 Verification 70
4.3 Floating Composition 75
4.3.1 Computations of a Transition Module 75
4.3.2 Floating Temporal Logic 76
4.3.3 Verification 76

5 Real-time Composition **79**
5.1 Real-time Transition Modules 79
5.1.1 The Environment Transition 80
5.1.2 Computations of a Transition Module 80
5.1.3 Parallel Composition 81
5.2 Metric Temporal Logic 82
5.2.1 A Decision Procedure 83
5.2.2 A Deductive System 86
5.3 Verification 90
5.3.1 The Environment Restriction 90
5.3.2 Global Verification 90
5.3.3 Compositional Verification 97
5.3.4 Bounded Response for Fischer's Algorithm 98

6 Conclusions **102**

Bibliography **103**

List of Figures

2.1 A fully-labeled program. 7
2.2 Inclusion relation among property classes. 13

3.1 A simple program. 20
3.2 Peterson's mutual exclusion algorithm. 26
3.3 A resource allocator system. 28
3.4 Program `PING-PONG-PING`. 34
3.5 Resource allocation by bounded asynchronous message passing. 41
3.6 Program `SYNC-PING-PONG`. 46

4.1 Anchored sequential composition. 49
4.2 Floating sequential composition. 50
4.3 An informal example. 51
4.4 Showing that $(\sigma, j) \models p \, \mathcal{F} \, q$. 57
4.5 Program `SIMPLE-SEQ`. 71
4.6 Program `DECREASE`. 73

5.1 Fischer's mutual exclusion algorithm. 92
5.2 A verification diagram. 99

Chapter 1

Introduction

This thesis presents a compositional methodology for the verification of reactive and real-time systems. Given a temporal logic specification for some system of interest, the correctness of that system is established by proving that each system component satisfies some appropriate specification. In turn, the correctness of each system component may be established by proving the appropriate specifications for its components. The advantages of compositional verification are clear. Each system component is both smaller and simpler than the system itself. Furthermore, the application of compositional techniques often provides greater insight into the interaction among system components than is provided by global techniques.

Compositional methods for sequential systems have been known for some time, but have yet to be adequately developed for concurrent systems, or, more generally, for reactive and real-time systems. Even though temporal logic has been widely used as a specification language for reactive systems since its introduction [Pnu77], the temporal framework has often been criticized because of its global nature.

Finally, although not considered in this thesis, a compositional proof system suggests a systematic strategy for the *development* of reactive systems, and may even lead to techniques for automatic or computer-assisted synthesis of reactive modules. Given a specification for a reactive system, a strategy for development is to decompose the global specification into *modular specifications*, i.e., specifications for each component. The compositional proof system may be used to ensure that the modular specifications imply the global specification.

This thesis is organized as follows. Chapter 2 reviews the temporal framework underlying this work, including transition systems, temporal logic, and the safety-progress classification of temporal properties. A new syntactic characterization of each property class is introduced, allowing an appropriate choice of proof rules for the verification of a given specification. Finally, suitable proof rules for each property class are presented. Chapter 3 introduces *transition modules* which, in contrast to transition systems, may be composed and may interact with the environment. The parallel composition of transition modules is defined, and shared variable, asynchronous and synchronous message passing communication paradigms are considered. Explicit proof rules for proving liveness properties are introduced. Sequential composition and iteration is introduced in Chapter 4. Sequential composition is treated quite differently in the anchored temporal semantics as compared to the floating temporal semantics, and each of these is considered in turn. Finally, Chapter 5 presents the natural extension of this methodology to real-time, and Chapter 6 considers a few directions for future research.

Chapter 2

The Global Framework

This chapter briefly reviews a methodology for the global specification and verification of complete systems. Each system is considered a monolithic entity, in the sense that no attempt is made to separately analyze the behavior of individual system components. In addition, a simple programming language is defined for describing small examples.

The material in this chapter serves as the foundation for the compositional framework of later chapters. As the analysis of a given system is reduced to the analysis of smaller and smaller system components, a point is reached at which further reduction is neither desirable nor feasible. At this point, each component is treated as a complete system, and the verification task for each component is carried out using global techniques.

The framework presented in this chapter is more thoroughly developed in [MP92] and [MP91].

2.1 Fair Transition Systems

Each system is modeled as a fair transition system, consisting primarily of an initial state predicate and a next-state relation. The next-state relation is defined by a set of transitions $\mathcal{T}$, which are predicates on pairs of states s and s', such that s' may follow s if and only if some transition $\tau \in \mathcal{T}$ holds on s and s'.

Each possible computation of the system is an infinite sequence of states such that the first state satisfies the initial state predicate, each pair of adjoining states belongs to the next-state relation, and no transition is continuously neglected.

More formally, a *fair transition system* S consists of the following components:

- V : A finite set of *state variables*. Each variable is associated with a non-empty domain over which it ranges.

  An interpretation over V is a V*-state*, or simply a *state* when the set of variables V is understood.

- $\mathcal{T}$: A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is an assertion of the form

  $$\tau : En(\tau) \wedge (\overline{y}' = \overline{e})$$

  which relates the values of the state variables in a state s to their values in a successor state s'. A state s' is a τ*-successor* of s if

  $$\langle s, s' \rangle \models \tau$$

  where $\langle s, s' \rangle$ is the joint interpretation which interprets x as $s[x]$, and interprets x' as $s'[x]$.

  A transition τ is *enabled* on s if the assertion $En(\tau)$ holds on s. A set of transitions T is enabled, i.e., $En(T)$ holds, if some transition in T is enabled.

- Θ : An *initial condition*, i.e., a satisfiable assertion characterizing the initial states of the system. A state satisfying Θ is an *initial state*.

2.1.1 The Idling Transition

The *idling transition* is defined:

$$\tau_I : \bigwedge_{u \in V} (u' = u)$$

A state s' is a τ_I*-successor* of s iff s and s' agree on each variable in V. Note that τ_I is not a member of $\mathcal{T}$, i.e., $\tau_I \notin \mathcal{T}$.

2.1.2 Computations of a Fair Transition System

An infinite sequence of V-states $\sigma : s_0, s_1, \ldots$ is defined to be a *computation* of the fair transition system S if it satisfies the following requirements:

- *Initiation* The state s_0 is an initial state.
- *Consecution* For every j, where $0 \leq j$:
  - either the state s_{j+1} is a τ-successor of the state s_j for some $\tau \in \mathcal{T}$, i.e., transition τ is *taken* at position j in σ,
  - or the state s_{j+1} is a τ_I-successor of the state s_j. In this case, an *idling step* was taken at position j in σ.
- *Justice* For each $\tau \in \mathcal{T}$, it is not the case that τ is continually enabled beyond some point in σ but taken at only finitely many positions in σ.

The set of computations of a fair transition system S is denoted $Comp(S)$.

2.2 A Simple Programming Language

A *program* consists of an optional name, a declaration section, and a body. A *declaration section* is a list of *declarations*, each of which consists of a list of variables and a type (a domain for the specified variables). A predicate characterizing suitable initial values may optionally be provided. The body of a program is a statement, where a statement in the language is one of the following:

- **skip**

  This statement does nothing and is always enabled.

- $\overline{y} := \overline{e}$

  This is an *assignment* statement, where $\overline{y}$ is a list of variables and $\overline{e}$ is a list of expressions of the same length and corresponding types. This statement is always enabled.

- **await** c

  This statement waits until the (boolean) *guard* c becomes true, at which point it terminates. This statement is enabled if the guard c is true.

- **if** c **then** S_1 **else** S_2

  This is a *conditional* statement, executing statement S_1 if the (boolean) *condition* c is true, and executing the statement S_2 otherwise. This statement is always enabled.

- **while** c **do** S_1

  This statement evaluates the boolean expression c. If the value of c is false, the statement terminates. Otherwise, the statement S_1 is executed. When and if S_1 terminates, this statement repeats itself. This statement is always enabled.

- $S_1; S_2$

  This is a *concatenation* statement. First statement S_1 is executed, and then S_2 is executed. This statement is enabled if S_1 is enabled.

  The *multiple concatenation* statement is given by $S_1; S_2; \ldots; S_n$.

- S_1 **or** S_2

  This is a *selection* statement. If both S_1 and S_2 are enabled, then one of them is nondeterministically selected for execution. Otherwise, whichever statement S_1 or S_2 is enabled is executed. This statement is enabled if either S_1 or S_2 is enabled.

  The *multiple selection* statement is given by S_1 **or** S_2 **or** $\ldots$ **or** S_n.

- $S_1 || S_2$

  This is a *cooperation* statement. The execution of S_1 and S_2 is interleaved. This statement is always enabled.

  The *multiple cooperation* statement is given by $S_1 || S_2 || \ldots || S_n$.

2.2.1 Statement Labels

Each statement of a program is preceded and followed by a label. Figure 2.1 gives an example of a (fully-labeled) program.

$$
\begin{array}{rl} b: & \textbf{boolean} \\ x: & \textbf{integer} \end{array}
$$

$$
\ell_0: \left[[\ell_1:\ b := \mathrm{F};\ \widehat{\ell}_1:] \ \| \ \left[\begin{array}{l} \ell_2:\ \textbf{while}\ b\ \textbf{do} \\ \quad \ell_3: \begin{bmatrix} \ell_4:\ x := x+1;\ \widehat{\ell}_4: \\ \textbf{or} \\ \ell_5:\ x := x-1;\ \widehat{\ell}_5: \end{bmatrix} \widehat{\ell}_3: \\ \widehat{\ell}_2: \end{array}\right] \right]
$$

Figure 2.1: A fully-labeled program.

An equivalence relation $\sim_L$ over labels is defined inductively:

- ℓ: **skip**; $\widehat{\ell}$:
- ℓ: $\overline{y} := \overline{e}$; $\widehat{\ell}$:
- ℓ: **await** c; $\widehat{\ell}$:

  The *skip*, assignment, and *await* statements do not introduce any label equivalences.

- ℓ: [**if** c **then** ℓ_1: S_1; $\widehat{\ell}_1$: **else** ℓ_2:; S_2 $\widehat{\ell}_2$:] $\widehat{\ell}$:

  $$\widehat{\ell}_1 \sim_L \widehat{\ell}_2 \sim_L \widehat{\ell}$$

- ℓ: [**while** c **do** ℓ_1: S_1; $\widehat{\ell}_1$:] $\widehat{\ell}$:

  The *while* statement does not introduce any label equivalences.

- ℓ: [ℓ_1: S_1; $\widehat{\ell}_1$:; ℓ_2: S_2; $\widehat{\ell}_2$:; $\ldots$; ℓ_n: S_n; $\widehat{\ell}_n$:] $\widehat{\ell}$:

  $$\begin{array}{l} \ell \sim_L \ell_1 \\ \widehat{\ell}_i \sim_L \ell_{i+1} \quad \text{for } i = 1, \ldots, n-1 \\ \widehat{\ell}_n \sim_L \widehat{\ell} \end{array}$$

- ℓ: $[\ell_1$: S_1; $\widehat{\ell}_1$: **or** ℓ_2: S_2; $\widehat{\ell}_2$: **or** $\ldots$ **or** ℓ_n: S_n; $\widehat{\ell}_n$:$]$ $\widehat{\ell}$:

$$\begin{array}{ll} \ell \sim_L \ell_i & \text{for } i = 1, \ldots, n \\ \widehat{\ell} \sim_L \widehat{\ell}_i & \text{for } i = 1, \ldots, n \end{array}$$

- ℓ: $[\ell_1$: S_1; $\widehat{\ell}_1$: $||$ ℓ_2: S_2; $\widehat{\ell}_2$: $||$ $\ldots ||$ ℓ_n: S_n; $\widehat{\ell}_n$:$]$ $\widehat{\ell}$:

  The cooperation statement does not introduce any label equivalences.

The equivalence class of a label ℓ is denoted $[\ell]$. An equivalence class of labels is a *location*. Statement labels that are not mentioned later in the text are occasionally omitted.

2.2.2 Semantics of the Programming Language

The semantics of a program P are given by associating P with a fair transition system S^P, defined as follows.

Let ℓ: S; $\widehat{\ell}$: be the body of program P. Then the fair transition system S^P associated with P is given by:

- V consists of all the variables declared in P, as well as a new control variable π, ranging over sets of locations.

- $\Theta : ([\ell] \in \pi) \wedge \varphi$, where φ is the conjunction of initial value predicates in the declarations.

- $\mathcal{T}$ consists of the transitions of S, as defined below.

Then the set of computations of P, denoted $Comp(P)$, is exactly the set of computations $Comp(S^P)$.

2.2.3 Transitions of a Statement

Let π be a control variable ranging over sets of locations, and define the predicate

$$moves(\pi, \ell, \widehat{\ell}) : \quad ([\ell] \in \pi) \wedge (\pi' = \pi \cup \{[\widehat{\ell}]\} - \{[\ell]\})$$

where $moves(\pi, \ell, \widehat{\ell})$ holds if control is currently at ℓ and subsequently moves to $\widehat{\ell}$. The set of transitions corresponding to a statement is defined inductively.

- ℓ: **skip**; $\widehat{\ell}$:

  The only transition for this statement is:

  $$\tau_\ell : \quad moves(\pi, \ell, \widehat{\ell}) \ \wedge \bigwedge_{u \in V - \{\pi\}} (u' = u)$$

  In particular, no variable in V other than π may be modified by the transition τ_ℓ. Henceforth, conjuncts of the form $(u' = u)$ will not be explicitly stated in giving the semantics of each statement.

- ℓ: $\overline{y} := \overline{e}$; $\widehat{\ell}$:

  The only transition for this statement is:

  $$\tau_\ell : \quad moves(\pi, \ell, \widehat{\ell}) \ \wedge (\overline{y}' = \overline{e})$$

- ℓ: **await** c; $\widehat{\ell}$:

  The only transition for this statement is:

  $$\tau_\ell : \quad c \ \wedge \ moves(\pi, \ell, \widehat{\ell})$$

- ℓ: [**if** c **then** ℓ_1: S_1; $\widehat{\ell}_1$: **else** ℓ_2: S_2; $\widehat{\ell}_2$:] $\widehat{\ell}$:

  The transitions of this statement are the transitions of S_1 and S_2, as well as:

  $$\tau_\ell : \quad (c \ \wedge \ moves(\pi, \ell, \ell_1)) \vee (\neg c \ \wedge \ moves(\pi, \ell, \ell_2))$$

- ℓ: [**while** c **do** ℓ': S'; $\widehat{\ell'}$:] $\widehat{\ell}$:

  The transitions of this statement are the transitions of S', as well as:

  $$\begin{aligned} \tau_\ell &: \quad (c \ \wedge \ moves(\pi, \ell, \ell')) \vee (\neg c \ \wedge \ moves(\pi, \ell, \widehat{\ell})) \\ \tau_{\widehat{\ell'}} &: \quad (c \ \wedge \ moves(\pi, \widehat{\ell'}, \ell')) \vee (\neg c \ \wedge \ moves(\pi, \widehat{\ell'}, \widehat{\ell})) \end{aligned}$$

- ℓ: $[\ell_1$: S_1; $\widehat{\ell}_1$:; ℓ_2: S_2; $\widehat{\ell}_2$:; $\ldots$; ℓ_n: S_n; $\widehat{\ell}_n$:$]$ $\widehat{\ell}$:

  The transitions of this statement are the transitions of each S_i.

- ℓ: $[\ell_1$: S_1; $\widehat{\ell}_1$: **or** ℓ_2: S_2; $\widehat{\ell}_2$: **or** $\ldots$ **or** ℓ_n: S_n; $\widehat{\ell}_n$:$]$ $\widehat{\ell}$:

  The transitions of this statement are the transitions of each S_i.

- ℓ: $[\ell_1$: S_1; $\widehat{\ell}_1$: $||$ ℓ_2: S_2; $\widehat{\ell}_2$: $||$ $\ldots$ $||$ ℓ_n: S_n; $\widehat{\ell}_n$:$]$ $\widehat{\ell}$:

  The transitions of this statement are the transitions of each S_i, as well as:

$$\begin{aligned} \tau_\ell^E &: \quad ([\ell] \in \pi) \wedge (\pi' = \pi \cup \{[\ell_1], \ldots, [\ell_n]\} - \{[\ell]\}) \\ \tau_\ell^X &: \quad (\{[\ell_1], \ldots, [\ell_n]\} \subseteq \pi) \wedge (\pi' = \pi \cup \{[\widehat{\ell}]\} - \{[\ell_1], \ldots, [\ell_n]\}) \end{aligned}$$

2.3 Temporal Logic

The language of temporal logic is used for specification. A *temporal formula* is constructed out of state formulas (equivalently, assertions), the boolean operators $\neg$ and $\vee$, and the following temporal operators:

$\bigcirc$ — Next	$\mathcal{U}$ — Until
$\ominus$ — Previous	$\mathcal{S}$ — Since

A *model* for a temporal formula p is an infinite sequence of $\mathcal{V}$-states $\sigma : s_0, s_1, \ldots$ where $\mathcal{V}$ includes at least the variables appearing in p. The set of variables $\mathcal{V}$ is partitioned into rigid and flexible variables. A *flexible variable* may have different values in different states, whereas a *rigid variable* must have the same value in every state of a model.

Given a model $\sigma : s_0, s_1, \ldots$ and a temporal formula p, $(\sigma, j) \vDash p$ denotes that p holds at position j in σ. For a state formula p,

$$(\sigma, j) \vDash p \Longleftrightarrow s_j \vDash p$$

That is, p is evaluated locally, using the interpretation given by s_j. The state s_j is a *p-state* if p holds on s_j.

$$
\begin{array}{lcl}
(\sigma, j) \vDash \neg p & \Longleftrightarrow & (\sigma, j) \nvDash p \\
(\sigma, j) \vDash p \vee q & \Longleftrightarrow & (\sigma, j) \vDash p \text{ or } (\sigma, j) \vDash q \\
(\sigma, j) \vDash \bigcirc p & \Longleftrightarrow & (\sigma, j+1) \vDash p \\
(\sigma, j) \vDash p \, \mathcal{U} \, q & \Longleftrightarrow & \text{for some } k,\ j \leq k,\ (\sigma, k) \vDash q \\
 & & \text{and for every } i,\ j \leq i < k,\ (\sigma, i) \vDash p \\
(\sigma, j) \vDash \ominus p & \Longleftrightarrow & j \geq 1 \text{ and } (\sigma, j-1) \vDash p \\
(\sigma, j) \vDash p \, \mathcal{S} \, q & \Longleftrightarrow & \text{for some } k,\ 0 \leq k \leq j,\ (\sigma, k) \vDash q \\
 & & \text{and for every } i,\ k < i \leq j,\ (\sigma, i) \vDash p
\end{array}
$$

Additional temporal operators can be defined as follows:

$$
\begin{array}{ll}
\Diamond p = \mathit{true} \, \mathcal{U} \, p & \text{— Eventually} \\
\Box p = \neg \Diamond \neg p & \text{— Henceforth} \\
p \, \mathcal{W} \, q = \Box p \vee (p \, \mathcal{U} \, q) & \text{— Waiting-for} \\
\diamondminus p = \mathit{true} \, \mathcal{S} \, p & \text{— Sometime in the past} \\
\boxminus p = \neg \diamondminus \neg p & \text{— Always in the past} \\
p \, \mathcal{B} \, q = \boxminus p \vee (p \, \mathcal{S} \, q) & \text{— Back-to} \\
\widetilde{\ominus} p = \neg \ominus \neg p & \text{— Weak Previous}
\end{array}
$$

Another useful operator is the *entailment* operator, defined by:

$$p \Rightarrow q \Longleftrightarrow \Box(p \rightarrow q)$$

The operators $\bigcirc$, $\mathcal{U}$, and the operators derived from them are the *future* operators; $\ominus$, $\mathcal{S}$, and the operators derived from them are the *past* operators. A formula that contains no future operators is called a *past formula.* A formula that contains no past operators is called a *future formula.* The operators $\bigcirc$ and $\ominus$ are referred to as the *immediate* operators.

A temporal formula p *holds* on a model σ, denoted $\sigma \vDash p$, if p holds at the first position of σ, i.e., $(\sigma, 0) \vDash p$. A formula p is *satisfiable* if it holds on some model; it is *valid*, denoted $\vDash p$, if it holds on all models. Formulas p and q are *equivalent*, denoted $p \sim q$, if $p \leftrightarrow q$ is valid.

A temporal formula p is *S-valid* (equivalently, *valid over S*), denoted $S \vDash p$, if for all computations σ of S, $\sigma \vDash p$.

2.3.1 Classification of Formulas

A complete proof system for program verification may be obtained by identifying a suitable classification of properties and then providing proof rules that are complete for each class. The *safety-progress* hierarchy [CMP92] is one such classification, and will serve as the basis for the global as well as the compositional proof systems to be presented later.

Canonical Formulas

The safety-progress hierarchy can be characterized by the types of formulas needed to express properties in a given class. For each property class $\kappa \in \{$ *safety*, *guarantee*, *obligation*, *response*, *persistence*, *reactivity* $\}$, a formula of φ specifies a κ property if and only if φ is equivalent to a canonical κ formula, defined as follows:

- A *canonical safety formula* is a formula of the form:

$$\Box\, p$$

- A *canonical guarantee formula* is a formula of the form:

$$\Diamond\, p$$

- A *canonical obligation formula* is a formula of the form:

$$\bigwedge_{i=1}^{m} (\Box\, p_i \;\vee\; \Diamond\, q_i)$$

- A *canonical response formula* is a formula of the form:

$$\Box \Diamond\, p$$

- A *canonical persistence formula* is a formula of the form:

$$\Diamond \Box\, p$$

- A *canonical reactivity formula* is a formula of the form:

$$\bigwedge_{i=1}^{m} (\Box \Diamond p_i \ \vee \ \Diamond \Box q_i)$$

Every formula is equivalent to some canonical reactivity formula, i.e., the class of reactivity properties is maximal.

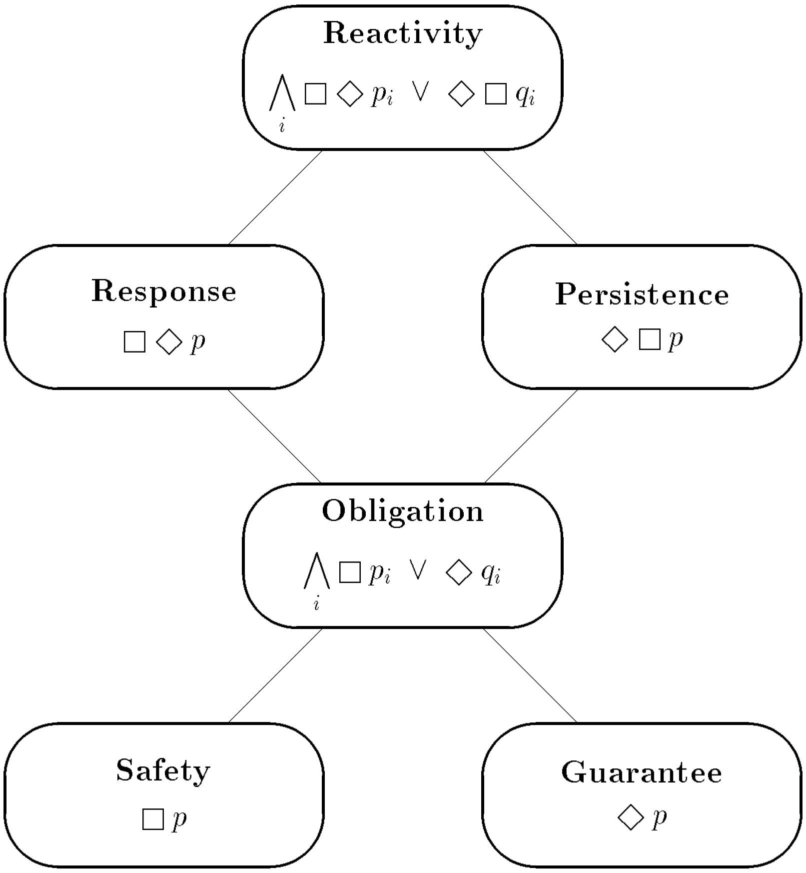

Figure 2.2: Inclusion relation among property classes.

The inclusion relation among the classes is presented in Figure 2.2.

Standard Formulas

The temporal characterization based on canonical formulas relies on boolean combinations of formulas of the form $\Box\, p$ and $\Box \Diamond\, p$, where p is an arbitrary *past* formula. This restricted form avoids the use of other future operators such as the *until* operator $\mathcal{U}$. It follows that, in order to determine the minimal class to which a formula belongs, it should be transformed first to canonical form. While this is always possible, the usual transformation through ω-automata may lead to an exponential blowup.

As a typical example, consider a system in which the event p should trigger the two responses r_1 and r_2, but r_1 should always weakly precede, i.e., precede or coincide with, r_2. The property stating that, from every occurrence of p, the next occurrence of r_2 must be weakly preceded by an occurrence of r_1 can be expressed by the temporal formula

$$p \Rightarrow (\neg r_2)\, \mathcal{W}\, r_1$$

which uses the *waiting-for* operator $\mathcal{W}$. Note that this formula does not state that r_2 will actually occur following p. It only states that if r_2 occurs, it must be weakly preceded by r_1. Consequently, it is a *safety* formula. To use the temporal characterization by canonical formulas, it must be transformed into canonical form. Indeed, the formula above is equivalent to

$$\Box(r_2 \rightarrow (\neg p)\, \mathcal{B}\, r_1)$$

which uses the *back-to* operator $\mathcal{B}$. This formula states that, going back from every occurrence of r_2, we must encounter an r_1 before we encounter a p. This form identifies the considered property as a *safety* property, since it it is of the form $\Box\, q$ for some past formula q. While the two formulas are equivalent, specifiers often prefer the use of *future* formulas and may find the need to transform to past-oriented specifications awkward and unnatural.

The *standard formulas* characterization of the safety-progress hierarchy remedies these drawbacks of the canonical characterization. The standard formulas characterization is applicable to arbitrary temporal formulas, including those that use the *until* and *unless* properties. Consequently, without any preliminary transformation, the presented characterization provides an upper bound on where a formula lies in the hierarchy.

Standard κ-formulas, for each class κ in the safety-progress hierarchy, are defined as follows.

a. standard *safety* formulas

- Every past formula is a standard safety formula.
- The negation of a standard guarantee formula is a standard safety formula.
- If p and q are standard safety formulas, then so are

$$p \vee q \qquad p \wedge q \qquad \bigcirc p \qquad \square p \qquad p \,\mathcal{W}\, q$$

b. standard *guarantee* formulas

- Every past formula is a standard guarantee formula.
- The negation of a standard safety formula is a standard guarantee formula.
- If p and q are standard guarantee formulas, then so are

$$p \vee q \qquad p \wedge q \qquad \bigcirc p \qquad \diamond p \qquad p \,\mathcal{U}\, q$$

c. standard *obligation* formulas

- Every standard safety and standard guarantee formula is a standard obligation formula.
- If p and q are standard obligation formulas, then so are

$$\neg p \qquad p \vee q \qquad p \wedge q \qquad \bigcirc p$$

- If p is a standard obligation formula and q is a standard guarantee formula, then the following is a standard obligation formula.

$$p \,\mathcal{U}\, q$$

- If p is a standard safety formula and q is a standard obligation formula, then the following is a standard obligation formula.

$$p \,\mathcal{W}\, q$$

d. standard *response* formulas

- Every standard safety and standard guarantee formula is a standard response formula.
- The negation of a standard persistence formula is a standard response formula.
- If p and q are standard response formulas, then so are

  $$p \vee q \qquad p \wedge q \qquad \bigcirc p \qquad \Box p \qquad p \,\mathcal{W}\, q$$

- If p is a standard response formula and q is a standard guarantee formula, then the following is a standard response formula.

  $$p \,\mathcal{U}\, q$$

e. standard *persistence* formulas

- Every standard safety and standard guarantee formula is a standard persistence formula.
- The negation of a standard response formula is a standard persistence formula.
- If p and q are standard persistence formulas, then so are

  $$p \vee q \qquad p \wedge q \qquad \bigcirc p \qquad \Diamond p \qquad p \,\mathcal{U}\, q$$

- If p is a standard safety formula and q is a standard persistence formula, then the following is a standard persistence formula.

  $$p \,\mathcal{W}\, q$$

f. If p and q are standard κ-formulas, where κ is one of safety, guarantee, obligation, response, or persistence, then the following are standard κ-formulas.

$$\circleddash p \qquad \diamondminus p \qquad \boxminus p \qquad p \,\mathcal{S}\, q \qquad p \,\mathcal{B}\, q$$

The definition does not mention the *reactivity* class since every temporal formula is a reactivity formula.

In the temporal hierarchy, every obligation formula is both a response formula and a persistence formula; although this is not stated explicitly in the above definition, it is clear that every standard obligation formula is both a standard response formula and a standard persistence formula.

2.4 Verification

2.4.1 Verification Conditions

For a transition $\tau \in \mathcal{T}$ and past formulas p and q, the *verification condition* of τ relative to p and q, denoted $\{p\}\tau\{q\}$, is the entailment

$$(\tau \wedge p) \Rightarrow q'$$

where V is the set of variables of the transition module containing τ. The primed version of a past formula q' is a past formula over $V \cup V'$ such that q' holds at a position iff q holds at the subsequent position, obtained inductively as follows:

$$\begin{aligned} (\ominus p)' &= p \\ (p \,\mathcal{S}\, q)' &= q' \vee (p' \wedge p \,\mathcal{S}\, q) \end{aligned}$$

The verification condition $\{p\}T\{q\}$ for a set of transitions T is given by $\bigwedge_{\tau \in T}\{p\}\tau\{q\}$.

2.4.2 Proof Rules for Safety Properties

The following proof rule is complete for safety properties [MP91].

SAFE

For a past formula φ:

$$\begin{array}{ll} \text{S1.} & \Theta \Rightarrow \varphi \\ \text{S2.} & \varphi \Rightarrow p \\ \text{S3.} & \{\varphi\}\mathcal{T}\{\varphi\} \\ \hline & \Box\, p \end{array}$$

2.4.3 Proof Rules for Response Properties

The following proof rule is complete for response properties [MP91].

RESP

For past formulas $\varphi = \varphi_1 \vee \cdots \vee \varphi_K$, transitions $\tau_1, \ldots, \tau_K$, well-founded domain $(A, \prec)$ and measure δ:

$$
\begin{array}{ll}
\text{R1.} & p \Rightarrow (q \vee \varphi) \\
\text{R2.} & \{\varphi_i \wedge \delta = \alpha\}\mathcal{T}\{q \vee (\varphi \wedge \delta \prec \alpha) \vee (\varphi_i \wedge \delta = \alpha)\} \\
\text{R3.} & \{\varphi_i \wedge \delta = \alpha\}\tau_i\{q \vee (\varphi \wedge \delta \prec \alpha)\} \\
\text{R4.} & \varphi_i \Rightarrow En(\tau_i) \\
\hline
 & p \Rightarrow \Diamond q
\end{array}
$$

2.4.4 Proof Rules for Reactivity Properties

The following proof rule reduces reactivity properties to simpler properties. It is complete for reactivity properties [MP91].

REAC

For a past formula φ and a well-founded domain $(A, \prec)$ with measure δ:

$$
\begin{array}{ll}
\text{R1.} & p \Rightarrow \varphi \mathcal{W} q \\
\text{R2.} & (\varphi \wedge \delta = \alpha) \Rightarrow (\delta \preceq \alpha) \mathcal{W} q \\
\text{R3.} & (\varphi \wedge r \wedge \delta = \alpha) \Rightarrow \Diamond(q \vee \delta \prec \alpha) \\
\hline
 & (p \wedge \Box \Diamond r) \Rightarrow \Diamond q
\end{array}
$$

Chapter 3

Parallel Composition

The fair transition systems model presented in Chapter 2 is sufficient for the global analysis of closed systems. In other words, the entire system must be available for analysis, and the system may not interact with the environment. However, the fair transition systems model is not amenable to compositional reasoning, in which the correctness of a system is derived from the correctness of its components. The fair transition systems model is also inadequate for modeling open systems, in which the environment may affect the behavior of a system.

The transition modules model introduced below supports the verification of large systems formed by the parallel composition of smaller components. A more general definition of transition modules is presented in the next chapter to model sequential composition and iteration.

3.1 Transition Modules

The computations of a transition system do not allow any interference by the environment. For instance, the computations of a transition system modeling M, presented in Figure 3.1, are all of the form

$$\langle at\_\ell_0, 0\rangle \xrightarrow{\tau_I} \cdots \xrightarrow{\tau_I} \langle at\_\ell_0, 0\rangle \xrightarrow{\tau_{\ell_0}} \langle at\_\ell_1, 1\rangle \xrightarrow{\tau_I} \cdots$$

$$M :: \begin{bmatrix} x : \textbf{integer where } x = 0 \\ \\ \ell_0\text{:} \quad x := x + 1 \\ \ell_1\text{:} \end{bmatrix}$$

Figure 3.1: A simple program.

where τ_I is the idling transition. The computations of a transition module, however, are open to interference by the environment. In particular, the computations of a transition module modeling M allow the environment, represented by the transition τ_E, to arbitrarily modify x:

$$\langle at\_\ell_0, 0\rangle \xrightarrow{\tau_E} \langle at\_\ell_0, m\rangle \xrightarrow{\tau_E} \cdots \xrightarrow{\tau_E} \langle at\_\ell_0, n\rangle \xrightarrow{\tau_{\ell_0}} \langle at\_\ell_1, n+1\rangle \xrightarrow{\tau_E} \cdots$$

If the behavior of the environment is entirely unconstrained, there is little that can usefully be said about the computations of M. Typically, however, the environment behaves more reasonably, and it is possible to determine that:

> If the environment behaves "reasonably,"
> then M satisfies some specification φ.

For instance, if the environment never decreases the value of x, then the value of x will eventually be greater than 0:

$$(\mathit{taken}(\tau_E) \Rightarrow (x' \geq x)) \quad \rightarrow \quad \Diamond(x > 0)$$

The antecedent $\mathit{taken}(\tau_E) \Rightarrow (x' \geq x)$ characterizes the environment of M, stating that when an environment step is taken, the value of x in the next state is greater or equal to the value of x in the current state. In this case, either M or the environment of M may set $x > 0$.

A transition module is typically a component of some larger system, so the state of the transition module is only part of the larger system state. Therefore, it is reasonable to assume that the variables V of the transition module are contained in some larger set of variables $\mathcal{V}$, which describe the state of the system, without explicitly stating what $\mathcal{V}$ contains.

A *transition module* M consists of the following components:

- $V \subseteq \mathcal{V}$: A finite set of variables, partitioned into private variables V^p and shared variables V^s.
- $\mathcal{T}$: A finite set of *transitions*, i.e., assertions over V and V'. A state s' is a τ-*successor* of s if:

  $$\langle s, s' \rangle \models \tau \wedge \bigwedge_{u \notin V} (u' = u)$$

  For every τ-successor s' of a state s, where $\tau \in \mathcal{T}$, there must be some $u \in V^p$ such that $s'[u] \neq s[u]$.
- Θ: An *initial condition*, i.e., an assertion over V characterizing the initial states of the transition module.

3.1.1 The Environment Transition

The environment transition τ_E is intended to capture every possible behavior of the environment of M. Specifically, a state s' is a τ_E-successor of s if s and s' agree on the private variables of M:

$$\langle s, s' \rangle \models \bigwedge_{u \in V^p} (u' = u)$$

In other words, the environment may exhibit arbitrary behavior, except that it may not modify any private variable of M.

3.1.2 Computations of a Transition Module

An infinite sequence of $\mathcal{V}$-states $\sigma : s_0, s_1, \ldots$ is defined to be a computation of M if it satisfies the following requirements:

- *Initiation* The state s_0 is an initial state.
- *Consecution* For every j, where $0 \leq j$:

  - either the state s_{j+1} is a τ-successor of the state s_j for some $\tau \in \mathcal{T}$, i.e., transition τ was *taken* at position j in σ,
  - or the state s_{j+1} is a τ_E-successor of state s_j, i.e., an *environment step* is taken at position j in σ.

- *Justice* For each $\tau \in \mathcal{T}$, it is not the case that τ is continually enabled beyond some point in σ but taken at only finitely many positions in σ.

The set of computations of M is denoted $Comp(M)$.

3.1.3 Parallel Composition

Transition modules M_1 and M_2 may be composed if the private variables of each module are not variables of the other module. Furthermore, the initial conditions of M_1 and M_2 must be compatible. Specifically:

- $V_1^p \cap V_2 = \emptyset$
- $V_2^p \cap V_1 = \emptyset$
- $\Theta_1 \wedge \Theta_2$ is satisfiable

The parallel composition of M_1 and M_2 yields the transition module M, defined as follows:

- $V = V_1 \cup V_2$

$$V^p = V_1^p \cup V_2^p$$
$$V^s = V_1^s \cup V_2^s$$

- $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$
- $\Theta = \Theta_1 \wedge \Theta_2$

Theorem 3.1.1 $Comp(M) = Comp(M_1) \cap Comp(M_2)$

Proof:

Let σ be a computation of M. The initiation requirement for M_1 is obviously satisfied. At any position j, either the environment transition or some transition of M is taken. The environment transition of M may not modify any private variable of M_1, so any environment step of M is also an environment step of M_1. Since $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, if some transition of M is taken at j, then either a transition of M_1 or M_2 is taken at j. Since no transition of M_2 may modify any private variable of M_1, a step taken by a transition of M_2 may correspond to an environment step of M_1. It follows that the consecution requirement for M_1 is satisfied. The justice requirement is also satisfied, since any transition of M_1 that is forever enabled but not taken would violate the justice requirement of M. By symmetry, σ is likewise a computation of M_2.

Conversely, let σ be a computation of both M_1 and M_2. Clearly, the initiation requirement for M is satisfied. At each position j, either a transition of M_1 or M_2 was taken, in which case the same transition belonging to M was taken, or an environment step was taken by both M_1 and M_2. Then no private variable of M_1 or M_2 could have been changed, so an environment step was also taken by M. The justice requirement is also satisfied, since any transition of M that is forever enabled but not taken would violate the justice requirement of either M_1 or M_2. Therefore, every computation of both M_1 and M_2 is also a computation of M. ∎

3.2 The Environment Restriction

The environment transition τ_E of a transition module M allows the environment to arbitrarily modify any variable not in V^p, i.e., any variable that is not a private variable of M. It is typically the case, however, that the environment of M behaves in a more restrained manner, and by characterizing the behavior of the actual environment it is possible to verify properties of M in a particular environment that do not hold for every environment.

An *environment restriction* *Env* is a formula of the form

$$Env: \quad taken(\tau_E) \Rightarrow \underbrace{\bigwedge_{i=1}^{m} (\varphi_i \rightarrow \psi_i')}_{\mathcal{E}}$$

where φ_i and ψ_i are assertions, stating that, if φ_i holds at a position j where an environment step is taken, then ψ_i must hold at the next position $j+1$. In other words, $\mathcal{E}$ defines an additional next-state relation that must be satisfied by every environment step.

Environment restrictions of the form $(y = k) \rightarrow (y' = k)$, where k is a static variable, state that the environment does not change the value of y, and may be abbreviated $(y' = y)$. Note that an environment restriction must be stuttering-insensitive, and may not refer to private variables of the module.

Typically, the specification for a transition module M will be of the form:

$$Env \rightarrow \varphi$$

Such a specification states that φ holds on every computation σ of M such that every environment step taken from a φ_i state leads to a ψ_i state.

3.3 Verification

Let M be a transition module formed by the parallel composition of transition modules $M_1, \ldots, M_N$. In order to verify that M satisfies some specification

$$Env \rightarrow \varphi$$

it suffices to find specifications

$$Env_i \rightarrow \varphi_i$$

for each module M_i, for $i = 1, \ldots, N$, such that:

(1) each module is correct: $M_i \models Env_i \rightarrow \varphi_i$

(2) the environment restriction of each module is satisfied, and

(3) the specifications φ_i imply φ: $\models (\bigwedge_{i=1}^{N} \varphi_i) \rightarrow \varphi$.

In order to check condition (2) for module M_j with environment restriction Env_j : $taken(\tau_{E_j}) \Rightarrow \mathcal{E}_j$, observe that $taken(\tau_{E_j})$ holds at position k of a computation of M if and only if $taken(M_i)$ holds at k, where $taken(M_i)$ holds if some transition of module M_i was taken, for $i \neq j$, or $taken(\tau_E)$ holds at k, i.e., the environment of M has taken a step. Therefore it suffices to prove the following conditions:

$$M_i \models Env_i \rightarrow (taken(M_i) \Rightarrow \textstyle\bigwedge_{j \neq i} \mathcal{E}_j)$$
$$\models \mathcal{E} \rightarrow \textstyle\bigwedge_i \mathcal{E}_i$$

Despite the apparent circularity, i.e., assuming that the environment restriction Env_i of M_i is satisfied while showing that M_i satisfies the environment restriction for each M_j, this line of reasoning is sound. In particular, each environment restriction is a safety property, and it has been established that such circular reasoning for safety properties is sound [AL89].

3.3.1 Proof Rules for Safety Properties

Let M denote the parallel composition of transition modules $M_1, M_2, \ldots, M_N$. The following simple proof rule is sound for any property φ. A similar rule will then be shown to be complete for safety properties. As usual, let:

$$Env : \quad taken(\tau_E) \Rightarrow \mathcal{E}$$
$$Env_i : \quad taken(\tau_{E_i}) \Rightarrow \mathcal{E}_i$$

PAR

For formulas $\varphi_1, \ldots, \varphi_N$ and environment restrictions $Env_1, \ldots, Env_N$:

P1.	$M_i \models Env_i \rightarrow \varphi_i$
P2.	$M_i \models Env_i \rightarrow (taken(M_i) \Rightarrow \bigwedge_{j \neq i} \mathcal{E}_j)$
P3.	$\models \mathcal{E} \rightarrow \bigwedge_i \mathcal{E}_i$
P4.	$\models (\bigwedge_i \varphi_i) \rightarrow \varphi$
	$M \models Env \rightarrow \varphi$

The first premise establishes that each module satisfies its specification. The second and third premises establish that the environment of each module behaves as expected. Finally, the fourth premise establishes that the specifications of each module, taken in conjunction, imply the specification of the composed system.

$$
\begin{array}{rl}
y_1, y_2 & : \textbf{boolean where } y_1 = \mathit{false}\,, y_2 = \mathit{false} \\
s & : \textbf{integer where } s = 1
\end{array}
$$

$$
\begin{array}{l}
M_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{ll} \ell_1\text{:} & \textbf{noncritical} \\ \ell_2\text{:} & (y_1, s) := (\mathit{true}\,, 1) \\ \ell_3\text{:} & \textbf{await } \neg y_2 \ \vee \ s \neq 1 \\ \ell_4\text{:} & \textbf{critical} \\ \ell_5\text{:} & y_1 := \mathit{false} \end{array} \right] \end{array} \right] \\
|| \\
M_2 :: \left[\begin{array}{l} m_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{ll} m_1\text{:} & \textbf{noncritical} \\ m_2\text{:} & (y_2, s) := (\mathit{true}\,, 2) \\ m_3\text{:} & \textbf{await } \neg y_1 \ \vee \ s \neq 2 \\ m_4\text{:} & \textbf{critical} \\ m_5\text{:} & y_2 := \mathit{false} \end{array} \right] \end{array} \right]
\end{array}
$$

Figure 3.2: Peterson's mutual exclusion algorithm.

Example: The safety requirement for Peterson's mutual exclusion algorithm, presented in Figure 3.2, may be stated as follows:

$$M \ \models \ \mathit{Env} \ \rightarrow \ \Box\neg(\mathit{at\_}\ell_4 \ \wedge \ \mathit{at\_}m_4)$$

where $M = M_1 || M_2$, and the environment restriction Env

$$
\begin{array}{rl}
\mathit{Env} : & \mathit{taken}(\tau_E) \Rightarrow \mathcal{E} \\
\mathcal{E} : & (y_1' = y_1) \ \wedge (y_2' = y_2) \ \wedge (s' = s)
\end{array}
$$

states that the environment of M does not modify y_1, y_2, or s.

A suitable specification for module M_1 may be given as follows:

$$M_1 \models Env_1 \rightarrow (at\_\ell_4 \Rightarrow (y_1 \wedge (\neg y_2 \vee s = 2))) \tag{3.1}$$

stating that y_1 must hold whenever M_1 is in its critical section, and either y_2 must be false or $s = 2$. An appropriate environment restriction for M_1 is obtained by choosing:

$$\begin{aligned} \mathcal{E}_1 : \quad & y_1' = y_1 \\ & \wedge (\neg y_2) \rightarrow (y_2' \rightarrow s' = 2) \\ & \wedge (s = 2) \rightarrow (s' = 2) \end{aligned}$$

requiring that the environment of M_1 does not modify y_1, does not change y_2 from *false* to *true* without also assigning $s = 2$, and does not assign s to any value other than 2. It is easy to see that every transition of M_2 satisfies $\mathcal{E}_1$, and also that the environment of $M = M_1 || M_2$ satisfies $\mathcal{E}_1$.

It is not hard to see that (3.1) holds. A similar specification may be given for M_2. Then, the conjunction

$$\begin{aligned} & at\_\ell_4 \Rightarrow (y_1 \wedge (\neg y_2 \vee s = 2)) \\ & \quad \wedge \\ & at\_m_4 \Rightarrow (y_2 \wedge (\neg y_1 \vee s = 1)) \end{aligned}$$

clearly implies

$$\square \neg (at\_\ell_4 \wedge at\_m_4)$$

as desired. ▪

Rule `PAR` is not complete for proving safety properties. In particular, consider the mutual exclusion property

$$\varphi : \quad i \neq j \Rightarrow \neg (at\_m_4[i] \wedge at\_m_4[j])$$

$$b[1],\ldots,b[N] : \textbf{boolean where } b[1]=\mathit{false},\ldots,b[N]=\mathit{false}$$

$$A :: \left[\begin{array}{l} \textbf{private } t : \textbf{integer} \\ \\ \ell_0:\ \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1:\ t := (t\%M)+1 \\ \ell_2:\ \textbf{if } b[t] \textbf{ then} \\ \quad \left[\begin{array}{l} \ell_3:\ b[t] := \mathrm{F} \\ \ell_4:\ \textbf{await } b[t] \\ \ell_5:\ b[t] := \mathrm{F}\end{array}\right]\end{array}\right]\end{array}\right] \;||\; C[i] :: \left[\begin{array}{l} m_0:\ \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1:\ \textbf{noncritical} \\ m_2:\ b[i] := \mathrm{T} \\ m_3:\ \textbf{await } \neg b[i] \\ m_4:\ \textbf{critical} \\ m_5:\ b[i] := \mathrm{T} \\ m_6:\ \textbf{await } \neg b[i]\end{array}\right]\end{array}\right]$$

Figure 3.3: A resource allocator system.

for the resource allocator system in Figure 3.3. Each customer process $C[i]$ communicates with the allocator process A through the shared variable $b[i]$. Initially, all the variables $b[i]$ are false. Customer $C[i]$ may request entrance to its critical section by setting $b[i]$ to true. When the allocator is ready to grant that request, it resets $b[i]$ to false. At this point, $C[i]$ may enter its critical section. The customer $C[i]$ eventually signals its departure from its critical section by again setting $b[i]$ to true, which the allocator acknowledges by resetting $b[i]$ to false. The declaration of t in the allocator process A states that t is a private variable of A, i.e., belongs to V_A^p. Private variables may be data variables as well as control variables.

It is not difficult to see that there are no specifications φ_A and $\varphi_{C[i]}$ and environment restrictions Env_A and $Env_{C[i]}$, for the allocator and customer processes respectively, satisfying the premises of rule `PAR` and establishing φ. Each component specification may rely only on the shared variables $b[i]$, and the critical observation pertaining to the correctness of the system is that

(1) each customer process $C[i]$ may enter its critical section only if $b[i]$ had been reset to false an odd number of times previously, and

(2) the allocator process ensures that there is at most one $b[i]$ satisfying this property.

However, neither statement is expressible in temporal logic [Wol81]. Although

$$\varphi_i : \quad at\_m_4[i] \Rightarrow (t = i)$$

would seem to be a promising specification for each customer, especially since the conjunction $\bigwedge_i \varphi_i$ implies the desired property φ, φ_i is not suitable as a specification for customer process $C[i]$ because it refers to the private variable t of the allocator process.

Consider for a moment the same resource allocator system with customer processes $C[i]$, but whose allocator process is unknown. Assume simply that the property φ continues to hold for the system. The allocator must be able to distinguish between states in which every customer is in its noncritical section, i.e., $at\_m_1[i]$ holds for every i, and states in which some customer is in its critical section, i.e., $at\_m_4[i]$ holds for some i, even though such states cannot be distinguished solely on the basis of the shared variables $b[i]$. In other words, there must be some predicate $crit[i]$ maintained by the allocator in order to distinguish between critical and noncritical states of customer process $C[i]$.

The predicate $crit[i]$ can distinguish between critical and noncritical states only by conforming to the protocol, i.e., by properly observing the sequence of values $b[i]$. A resulting verification strategy proceeds as follows:

(1) Assume there is some predicate $crit[i]$ such that, if $crit[i]$ "behaves" according to the protocol and $C[i]$ is in its critical section, then $crit[i]$ must be true.

(2) The allocator provides predicates $crit[i]$ and $crit[j]$, and ensures that they are mutually exclusive, for $i \neq j$.

The following environment restriction $Env_{C[i]}$ of customer process $C[i]$ requires the predicate $crit[i]$ to follow the established protocol:

$$\begin{aligned}
Env_{C[i]} : &\quad taken(\tau_{E_{C[i]}}) \Rightarrow \mathcal{E}_{C[i]} \\
\mathcal{E}_{C[i]} : &\quad (\neg b[i] \wedge \neg crit[i]) \rightarrow (\neg b[i]' \wedge \neg crit[i]') \\
&\quad \wedge (b[i] \wedge \neg crit[i]) \rightarrow (\neg b[i]' \equiv crit[i]') \\
&\quad \wedge (\neg b[i] \wedge crit[i]) \rightarrow (\neg b[i]' \wedge crit[i]') \\
&\quad \wedge (b[i] \wedge crit[i]) \rightarrow (\neg b[i]' \equiv \neg crit[i]')
\end{aligned}$$

The first conjunct states that, when the customer $C[i]$ is not in its critical section and $b[i]$ is *false*, i.e., the customer is not requesting entry to the critical section, the allocator should not change $b[i]$ or $crit[i]$. The second conjunct states that, when the customer $C[i]$ is not in its critical section and $b[i]$ is true, i.e., the customer is requesting entry to the critical section, the allocator may grant entry to the critical section, by resetting $b[i]$ to *false*, only if the predicate $crit[i]$ is also set to *true*, recording the fact that the customer $C[i]$ has been granted permission to enter. Similarly, the third conjunct states that, when the customer $C[i]$ is in its critical section and $b[i]$ is *false*, i.e., the customer is not departing its critical section, the allocator should not change $b[i]$ or $crit[i]$, and the final conjunct states that, when the customer $C[i]$ is in its critical section and $b[i]$ is true, i.e., the customer is signaling its departure from its critical section, the allocator may acknowledge the signal by resetting $b[i]$ to *false* only if the predicate $crit[i]$ is also reset to *false*.

Now it is possible to establish the following specification for customer process $C[i]$:

$$C[i] \ \vDash (\forall crit[i]) \left[Env_{C[i]} \ \rightarrow (at\_m_4[i] \Rightarrow \ crit[i]) \right]$$

Furthermore, taking the environment restriction of the allocator to be:

$$Env_A : \quad taken(\tau_{E_A}) \Rightarrow \underbrace{\bigwedge_i (b[i] \rightarrow b[i]')}_{\mathcal{E}_A}$$

stating that no customer may grant its own request or acknowledge its own release, it is possible to establish:

$$A \ \vDash (\exists crit[i]) \left[Env_A \rightarrow \Box\, \mathcal{E}_{C[i]} \right]$$

by taking $crit[i]$ to be:

$$crit[i] : \quad at\_\ell_{4,5} \ \wedge (t = i)$$

After a little more effort it is possible to conclude

$$A||C[i] \ \vDash \ Env \ \rightarrow \ \bigwedge_i (at\_m_4[i] \Rightarrow (at\_\ell_{4,5} \ \wedge (t = i)))$$

for the entire system, which clearly implies the desired conclusion:

$$A||C[i] \ \models \ Env \ \rightarrow (i \neq j \Rightarrow \neg(at\_m_4[i] \ \wedge \ at\_m_4[j]))$$

Rule `GPAR`, presented below, justifies this line of reasoning. Each component M_i must be shown to satisfy some specification $Env_i \rightarrow \varphi_i$, where Env_i and φ_i may refer to variables in $V_i \cup \{x_i^j \mid j \neq i\}$. The variables x_i^j are *auxiliary variables*. The predicate $crit[i]$ in the resource allocator system corresponds to $x^A_{C[i]}$, i.e., a variable representing some part of the global state that is not visible to customer $C[i]$. Similarly, each x_i^j represents some part of the global state that is not visible to process i. Note that the distinction between x_i^j and x_i^k is not relevant during the verification of each separate component, i.e., the correctness of process i does not depend on whether the hidden state represented by x_i^j and x_i^k belongs to process j or k, but is used only to simplify the proof rule. Of course, M denotes the parallel composition of the component modules M_i.

`GPAR`

Let x_i^j be new variables for each $j \neq i$. For specifications φ_i and environment restrictions Env_i, each over $V_i \cup \{x_i^j \mid j \neq i\}$, and terms A_i^j over V_j^p:

$$\begin{array}{ll}
\text{P1.} & M_i \ \models \ Env_i \ \rightarrow \ \varphi_i \\
\text{P2.} & M_i \ \models \ Env_i \ \rightarrow (taken(M_i) \Rightarrow \bigwedge_{j \neq i} \mathcal{E}_j[A_j^i/x_j^i]) \\
\text{P3.} & \models (\mathcal{E} \ \wedge \ \bigwedge x_j^{i\prime} = x_j^i) \rightarrow \bigwedge_i \mathcal{E}_i \\
\text{P4.} & \models (\bigwedge_i \varphi_i[A_i^j/x_i^j]) \ \rightarrow \ \varphi \\
\hline
 & M \models Env \ \rightarrow \ \varphi
\end{array}$$

Proof:

In order to establish soundness, observe that

$$\begin{array}{l}
M_i \ \models \ Env_i^* \ \rightarrow \ \varphi_i^* \\
M_i \ \models \ Env_i^* \ \rightarrow (taken(M_i) \Rightarrow \bigwedge_{j \neq i} \mathcal{E}_j^*) \\
\models \ \mathcal{E} \ \rightarrow \ \bigwedge_i \mathcal{E}_i^* \\
\models (\bigwedge_i \varphi_i^*) \ \rightarrow \ \varphi
\end{array}$$

follow from the premises of rule GPAR, where φ_i^*, $\mathcal{E}_i^*$, and Env_i^* are obtained from φ_i, $\mathcal{E}_i$, and Env_i by replacing variables x_i^j by the corresponding terms A_i^j, for each $j \neq i$. In particular, this holds because each x_i^j and the variables in each A_i^j are not in V_i.

Let σ be a computation of M satisfying Env. It is easy to show that σ is a computation of each M_i satisfying Env_i^*, and therefore also satisfying each φ_i^*. Otherwise let k be the first position such that, for some i, $(\sigma, k) \not\models (taken(\tau_{E_i}) \rightarrow \mathcal{E}_i^*)$. At position k in the computation σ of M, either $\mathcal{E}$ holds, in which case $\mathcal{E}_i^*$ holds, or some transition of M is taken. Since Env_i is false, it cannot be the case that some transition of M_i is taken or $taken(\tau_{E_i})$ would be false. Therefore a transition of M_j is taken at k, for some $j \neq i$. Observe that $(taken(\tau_{E_j}) \rightarrow \mathcal{E}_j^*)$ holds at every position up to and including k, $taken(M_j)$ holds at k, but $\mathcal{E}_i^*$ does not hold at k. It is easy to extend the computation to obtain a computation of M_j that does not satisfy $Env_j^* \rightarrow (taken(M_j) \Rightarrow \mathcal{E}_i^*$, which is a contradiction.

Rule GPAR is relatively complete for safety properties. Assume that each component M_i has exactly one private variable π_i. If this is not the case, π_i can be taken to be the tuple of private variables in V_i^p. Choose $\mathcal{E}_i$ to be the disjunction of $\mathcal{E} \wedge (x_i^{j\prime} = x_i^j)$ and all the transitions in each $\mathcal{T}_j$, for $j \neq i$, replacing π_j by x_i^j throughout. In particular, for a transition $\tau \in \mathcal{T}_j$, $\mathcal{E}_i$ includes the disjunct:

$$\tau[x_i^j/\pi_j] \ \wedge \bigwedge_{k \neq i, k \neq j} ((x_i^k)' = x_i^k) \ \wedge \bigwedge_{u \in V_i^s - V_j^s} (u' = u)$$

Each φ_i is taken to be φ, replacing π_j by x_i^j throughout.

It is easy to show that each of the premises is valid. For P1, observe that, from each computation of M_i satisfying Env_i, it is possible to construct a sequence of states satisfying the initiation and consecution conditions for computations of M, as well as the environment restriction Env. If φ specifies a safety property, then any computation of M_i satisfying Env_i but not φ_i yields a computation of M satisfying Env but not φ. For P2, $taken(M_i)$ implies $taken(\tau)$ for some $\tau \in \mathcal{T}_i$, and by construction $\mathcal{E}_j[\pi_i/x_j^i]$ is satisfied when τ is taken. The final two premises are trivial. ∎

With the addition of auxiliary variables to the environment restriction Env, as allowed by rule G-PAR, the environment is almost being treated as a full-fledged transition system executing in parallel with the transition module. Although such a duality would be elegant in principle, in practice it is simpler not to accord the environment equal status with the transition module. The substitution of terms A_i^j for auxiliary variables x_i^j implicitly determines a refinement mapping [Lam83] [LS83]. The existence of such mappings for a given (environment) specification is considered in [AL91]. In this thesis, the question of existence of refinement mappings is sidestepped by allowing each environment specification to be chosen in such a way that the refinement mapping is known to exist.

3.3.2 Proof Rules for Response Properties

In order to verify response properties, a stronger rule than GPAR may be required. Let M be a transition module composed of transition modules $M_1, \ldots, M_N$, and consider the following specification:

$$M \models Env \rightarrow (p \Rightarrow \Diamond q)$$

The global verification strategy is to identify some measure δ over a well-founded domain $(A, \prec)$ such that, starting from any p-state in a computation of M, the value of δ must repeatedly decrease until a q-state is reached. The compositional verification strategy requires identifying states, or more generally, histories, in which it is the responsibility of a given component to progress, i.e., to decrease the value of δ. In the following proof rule, premise PR1 states that, following a p-state, some φ_i must hold up to the occurrence of the next q state. Premise PR2 states that component M_i is responsible for making progress towards q whenever φ_i holds. Note that δ_i in PR2 refers to a well-founded measure over $(A_i, \prec_i)$ such that $\alpha_i \prec_i \beta_i$ implies $\alpha_i^* \prec \beta_i^*$, where $\alpha_i, \beta_i \in A_i$, and $\alpha_i^*, \beta_i^* \in A$ are obtained by replacing x_i^j by A_i^j in α_i and β_i, respectively, for $j \neq i$. These concepts are illustrated in the examples following the proof rule.

$\mathtt{PAR-RESP}$

Let x_i^j be new variables for each $j \neq i$. For past formulas φ_i and environment restrictions Env_i, each over $V_i \cup \{x_i^j \mid j \neq i\}$, terms A_i^j over V_j^p, and a well-founded domain $(A, \prec)$ with measure δ:

$$
\begin{array}{ll}
\text{PR1.} & M \ \models \ Env \rightarrow (p \Rightarrow (\bigvee_i \varphi_i^*)\, \mathcal{W}\, q) \\
\text{PR2.} & M_i \ \models \ Env_i \rightarrow ((\varphi_i \ \wedge \ \delta_i = \alpha_i) \Rightarrow \diamondsuit(q \ \vee \ \delta_i \prec \alpha_i)) \\
\text{PR3.} & M_i \ \models \ Env_i \rightarrow (taken(M_i) \Rightarrow \bigwedge_{j \neq i} \mathcal{E}_j[A_j^i / x_j^i]) \\
\text{PR4.} & \models (\mathcal{E} \ \wedge \ x_j^{i\,\prime} = x_j^i) \ \rightarrow \ \bigwedge_i \mathcal{E}_i \\
\hline
 & M \ \models \ Env \ \rightarrow (p \Rightarrow \diamondsuit\, q)
\end{array}
$$

Proof:

It is not difficult to see that rule PAR-RESP is sound and relatively complete. To establish completeness, environment restrictions may be chosen as described in the proof of rule GPAR. Choose formulas $\psi_1, \ldots, \psi_K$ and transitions $\tau_1, \ldots, \tau_K$ that would be appropriate for global verification of M, using rule RESP as presented in section 2.4. Then φ_i for each component M_i may be taken to be the disjunction of ψ_k such that τ_k is a transition of M_i, replacing private variables in ψ_k of components other than M_i by the appropriate x_i^j. ∎

$$x,\, y,\, z : \textbf{integer where } x = 0,\ y = 0$$

$$
M_1 :: \begin{bmatrix} \ell_0\colon & x := 1 \\ \ell_1\colon & \textbf{await } y = 1 \\ \ell_2\colon & x := 0 \end{bmatrix} \quad || \quad M_2 :: \begin{bmatrix} m_0\colon & \textbf{await } x = 1 \\ m_1\colon & y := 1 \\ m_2\colon & \textbf{await } x = 0 \\ m_3\colon & z := 1 \end{bmatrix}
$$

Figure 3.4: Program PING-PONG-PING.

Example: Rule PAR-RESP may be used to establish

$$Env \ \rightarrow (\Theta \Rightarrow \Diamond\, z = 1)$$

for program PING-PONG-PING in Figure 3.4, where Env states that the environment does not modify x, y or z. In particular, the premises of PAR-RESP are valid for:

$$\begin{aligned}
\mathcal{E}_1 &: \ (x' = x) \ \wedge (y = 1 \rightarrow y' = 1) \\
\varphi_1 &: \ (x = 0 \ \wedge \ y = 0) \ \vee (x = 1 \ \wedge \ y = 1)
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_2 &: \ (y' = y) \\
\varphi_2 &: \ (x = 1 \ \wedge \ y = 0) \ \vee (x = 0 \ \wedge \ y = 1)
\end{aligned}$$

$$\begin{aligned}
\delta : \ x = 0 \ \wedge \ y = 0 \ &\succ \ x = 1 \ \wedge \ y = 0 \\
&\succ \ x = 1 \ \wedge \ y = 1 \\
&\succ \ x = 0 \ \wedge \ y = 1
\end{aligned}$$

▪

Example: As a slightly more interesting example, observe that

$$Env \ \rightarrow (at\_m_2 \Rightarrow \Diamond\, at\_m_4) \tag{3.2}$$

is valid for Peterson's mutual exclusion algorithm, presented earlier. The following line of reasoning establishes the desired property:

From a state in which at_m_2 holds, $\varphi_{2,1} \vee \varphi_1 \vee \varphi_{2,2}$ must hold at least up to at_m_4, where

- $\varphi_{2,1}$: $at\_m_2 \wedge \neg y_2$ characterizes states in which process M_2 is responsible for proceeding to a state satisfying $\varphi_1 \vee \varphi_{2,2}$, and
- φ_1 : $at\_m_3 \wedge y_1 \wedge s = 2$ characterizes states in which process M_1 is responsible for proceeding through its critical section to a state satisfying $\varphi_{2,2}$, and

- $\varphi_{2,2}$: $at\_m_3 \wedge (\neg y_1 \vee s = 1)$ characterizes states in which process M_2 is responsible for proceeding to its critical section at_m_4.

The proof obligation of showing that M_1 proceeds from a φ_1 state to a $\varphi_{2,2}$ state requires the introduction of the auxiliary variable x_1^2, representing the hidden state at_m_3. The following environment restriction for M_1

$$Env_1 : \quad taken(\tau_{E_1}) \Rightarrow \underbrace{x_1^2 \wedge y_1 \wedge s = 2 \rightarrow x_1^{2\prime}}_{\mathcal{E}_1}$$

states that the environment of M_1 does not change the value of x_1^2, i.e., at_m_3, to false while $y_1 \wedge s = 2$ holds, and the following valid specification for M_1

$$Env_1 \rightarrow (x_1^2 \wedge y_1 \wedge s = 2) \Rightarrow \diamondsuit(x_1^2 \wedge (\neg y_1 \vee s \neq 2))$$

states that, if the environment behaves accordingly, then M_1 ensures progress as desired.

Therefore, in this example, δ may be taken to be:

$$\begin{array}{lrl} \delta : \quad \neg y_2 & \succ & at\_m_3 \wedge y_1 \wedge s = 2 \\ & \succ & at\_m_3 \wedge (\neg y_1 \vee s \neq 2) \end{array}$$

whereas the corresponding δ_1 would be:

$$\begin{array}{lrl} \delta : \quad \neg y_2 & \succ & x_1^2 \wedge y_1 \wedge s = 2 \\ & \succ & x_1^2 \wedge (\neg y_1 \vee s \neq 2) \end{array}$$

▟

3.3.3 Layered Decomposition

Rule PAR-RESP provides a certain structure to the task of verifying response properties. Specifically, it suffices to find appropriate formulas φ_i, environment restrictions Env_i, and well-founded measure δ satisfying the premises of rule PAR-RESP. An alternative, less-structured methodology is described in [MP92], where it is referred to as *layered decomposition.* Layered decomposition is the process of first, identifying properties of each component process that hold unconditionally, and second, introducing conditional properties of each process, i.e., properties that depend on unconditional properties or earlier conditional properties of the system.

However, since shared variables may be modified arbitrarily by the environment, which was not the case in [MP92], it is rarely the case that useful properties will hold unconditionally for a given module. Therefore, a slightly more general formulation of layered decomposition may be described as follows:

- Identify appropriate environment restrictions for each module,
- introduce "unconditional" properties, depending only on the environment restriction, for each module, and
- introduce conditional properties which may depend on both the unconditional properties or earlier conditional properties.

The proof obligations for layered decomposition are exactly those required as the premises of rule `PAR`, where the specification φ_i is taken to be the conjunction of the unconditional and conditional properties attributed to process i.

Example: A weaker environment restriction for M_1 may be used for establishing the property

$$Env \ \rightarrow (\Theta \Rightarrow \Diamond\, z = 1)$$

of program `PING-PONG-PING` when using layered decomposition. In particular, choose

$$\begin{array}{ll} \mathcal{E}_1 : & x' = x \\ \mathcal{E}_2 : & y' = y \end{array}$$

and observe the following "unconditional" property:

$$M_1 \ \models \ Env_1 \ \rightarrow \ \underbrace{\Diamond((x = 1)\, \mathcal{W}\, (y = 1))}_{\varphi_{1,1}}$$

The following conditional properties also hold:

$$
\begin{array}{lcl}
M_2 & \vDash & Env_2 \rightarrow \underbrace{\diamondsuit((x=1)\,\mathcal{W}\,(y=1)) \rightarrow \diamondsuit\,\square\, y=1}_{\varphi_{2,1}} \\
M_1 & \vDash & Env_1 \rightarrow \underbrace{\diamondsuit\,\square\, y=1 \rightarrow \diamondsuit\,\square\, x=0}_{\varphi_{1,2}} \\
M_2 & \vDash & Env_2 \rightarrow \underbrace{(\diamondsuit\,\square\, x=0 \ \wedge\ \diamondsuit\,\square\, y=1) \rightarrow \diamondsuit\, z=1}_{\varphi_{2,2}}
\end{array}
$$

Rule `PAR` may be applied, taking the component specifications φ_1 and φ_2 to be $\varphi_1 : \ \varphi_{1,1} \ \wedge \ \varphi_{1,2}$ and $\varphi_2 : \ \varphi_{2,1} \ \wedge \ \varphi_{2,2}$, respectively. ▟

3.3.4 Proof Rules for Reactivity Properties

The following proof rule reduces reactivity properties to simpler properties, which may then be verified compositionally. It is complete for reactivity properties [MP91].

`REAC`

For a past formula φ and a well-founded domain $(A, \prec)$ with measure δ:

$$
\begin{array}{ll}
\text{R1.} & p \ \Rightarrow \ \varphi\,\mathcal{W}\,q \\
\text{R2.} & (\varphi \ \wedge \ \delta=\alpha) \ \Rightarrow \ (\delta \preceq \alpha)\,\mathcal{W}\,q \\
\text{R3.} & (\varphi \ \wedge \ r \ \wedge \ \delta=\alpha) \ \Rightarrow \ \diamondsuit(q \ \vee \ \delta \prec \alpha) \\
\hline
 & (p \ \wedge \ \square\,\diamondsuit\, r) \Rightarrow \diamondsuit\, q
\end{array}
$$

3.3.5 Embedded Transition Modules

Compositional verification is a two-stage process. First, given a system composed of several modules, repeatedly apply composition rules (such as rule `PAR-RESP`) to reduce the verification task for the entire system to the task of verifying each of the smaller components. Second, when no further reduction is possible or desirable, each of the components must then be verified individually. The task of verifying that a transition module M meets its specification

$$M \ \vDash \ Env \rightarrow \varphi$$

may be reduced to verifying that the embedded transition module $[M, Env]$, defined below to be a transition system, satisfies φ.

Given a transition module M and an environment restriction Env, the *embedded transition module* $[M, Env]$ is defined to be the transition system $\widehat{M} = \langle \widehat{V}, \widehat{\mathcal{T}}, \widehat{\Theta} \rangle$:

$$\widehat{V} = V$$

$$\widehat{\mathcal{T}} = \mathcal{T} \cup \{\tau\}$$

$$\text{where } \tau : \quad \tau_E \ \wedge \ \mathcal{E}$$

$$\widehat{\Theta} = \Theta$$

If $\mathcal{E}$ refers to any variables not in V, they are also included in $\widehat{V}$ and, for each such variable u, the conjunct $(u' = u)$ is added to each $\tau \in \mathcal{T}$.

It is easy to see that the following theorem holds.

Theorem 3.3.1 $M \ \models \ Env \rightarrow \varphi$
iff
$[M, Env] \ \models \ \varphi$

3.4 Asynchronous Communication

It is not difficult to extend the current shared-variables framework to allow message-passing via bounded or unbounded asynchronous communication channels. Each channel is modeled by a variable representing a queue of messages; if the channel is bounded, then the queue cannot exceed the specified length.

The simple programming language of section 2.2 is extended by allowing channel declarations of the form

α : **channel** $[1 \ldots M]$
β : **channel** $[1 \ldots]$

where α and β are declared to be bounded and unbounded FIFO channels, respectively. The following communication statements are provided:

ℓ_s: $\alpha \Leftarrow e$; $\quad \widehat{\ell_s}$:
ℓ_r: $\alpha \Rightarrow x$; $\quad \widehat{\ell_r}$:

The *send statement*, labeled ℓ_s, places the value of expression e into the channel α. If α is a bounded channel of length M and already contains M messages, then the statement is not enabled; otherwise, the statement is enabled. The *receive statement*, labeled ℓ_r, removes the first message from the queue, if one exists, and stores the value in x. If there are no messages in the channel, then the statement is not enabled.

The following *communication transitions* are associated with ℓ_s and ℓ_r, for the case that α is an unbounded channel:

$$\begin{aligned}
\tau_{\ell_s} &: \ \mathit{moves}(\pi, \ell_s, \widehat{\ell_s}) \ \wedge \ \alpha' = \alpha \bullet e \\
\tau_{\ell_r} &: \ \mathit{moves}(\pi, \ell_r, \widehat{\ell_r}) \ \wedge \ |\alpha| > 0 \ \wedge \ \alpha' = \mathit{tail}(\alpha) \ \wedge \ x' = \mathit{head}(\alpha)
\end{aligned}$$

For the case that α is a bounded channel that can buffer at most M messages:

$$\begin{aligned}
\tau_{\ell_s} &: \ \mathit{moves}(\pi, \ell_s, \widehat{\ell_s}) \ \wedge \ |\alpha| < M \ \wedge \ \alpha' = \alpha \bullet e \\
\tau_{\ell_r} &: \ \mathit{moves}(\pi, \ell_r, \widehat{\ell_r}) \ \wedge \ |\alpha| > 0 \ \wedge \ \alpha' = \mathit{tail}(\alpha) \ \wedge \ x' = \mathit{head}(\alpha)
\end{aligned}$$

As usual, conjuncts of the form $(u' = u)$ have been omitted.

3.4.1 The Compassion Requirement

An additional fairness requirement must be placed on the computations of a transition module M modeling message-passing. Specifically, in order for a sequence of states σ to be a computation of M, it must satisfy the requirements of initiality, consecution, justice and:

- *Compassion* For each communication transition τ, it is not the case that τ is infinitely often enabled in σ but taken at only finitely many positions in σ.

Example: The resource allocator in Figure 3.5 satisfies the response property

$$\mathit{Env} \rightarrow \Box \Diamond (|\alpha| = 0) \tag{3.3}$$

where Env states that α and β are not changed by the environment. Each customer process $C[i]$ may enter its critical section only after sending a message on channel α. When it departs its critical section, it send a message on channel

β, and only then does the allocator clear both channels. Since each channel can hold at most one message, two customer processes cannot simultaneously enter their critical sections.

Rule PAR-RESP may be applied, where $p:\ \mathit{true}$ and $q:\ |\alpha| = 0$, taking:

$$\begin{aligned} \mathcal{E}_A &: \ (|\beta| = 1) \ \rightarrow (|\beta'| = 1) \\ \varphi_A &: \ |\alpha| = |\beta| = 1 \end{aligned}$$

$$\begin{aligned} \mathcal{E}_{C[i]} &: \ (|\alpha| = 1) \ \wedge (|\beta| = 0) \ \rightarrow (|\alpha'| = 1) \ \wedge (0 \le |\beta'| \le 1) \\ \varphi_{C[i]} &: \ (|\alpha| = 1) \ \wedge (|\beta| = 0) \ \wedge \ at\_m_{3,4}[i] \end{aligned}$$

$$\delta : \ (\textstyle\bigvee_i \varphi_{C[i]}) \ \succ \ \varphi_A$$

$\blacksquare$

$$\alpha, \beta : \textbf{channel}\,[1 \ldots 1] \ \textbf{where} \ \alpha = \Lambda, \beta = \Lambda$$

$$A :: \begin{bmatrix} \textbf{private } t : \textbf{integer} \\ \ell_0\text{:} \ \ \textbf{loop forever do} \\ \quad \begin{bmatrix} \ell_1\text{:} \ \ \beta \Rightarrow t \\ \ell_2\text{:} \ \ \alpha \Rightarrow t \end{bmatrix} \end{bmatrix} \quad || \quad C[i] :: \begin{bmatrix} m_0\text{:} \ \ \textbf{loop forever do} \\ \quad \begin{bmatrix} m_1\text{:} \ \ \textbf{noncritical} \\ m_2\text{:} \ \ \alpha \Leftarrow 0 \\ m_3\text{:} \ \ \textbf{critical} \\ m_4\text{:} \ \ \beta \Leftarrow 0 \end{bmatrix} \end{bmatrix}$$

Figure 3.5: Resource allocation by bounded asynchronous message passing.

3.4.2 Compassionate Proof Rules

With the introduction of the compassion requirement, rule PAR-RESP and rule REAC remain sound but are no longer complete. For instance, it is not possible to establish the accessibility property

$$Env \ \rightarrow (at\_m_2[i] \Rightarrow \Diamond \ at\_m_3[i])$$

for the resource allocator of Figure 3.5. The following proof rule may be applied instead. Premise PR1 states that every p-state is either also a q-state, or satisfies some φ_i. By premise PR2, φ_i must persist unless q is achieved or progress is made towards q. Premise PR3 ensures that there will be infinitely many ψ_i-states unless progress towards q is made, and PR4 states that, when φ_i holds and infinitely many ψ_i-states occur, then transition module M_1 guarantees progress towards q.

`PAR − CRESP`

Let x_i^j be new variables for each $j \neq i$. For past formulas φ_i, ψ_i and environment restrictions Env_i, each over $V_i \cup \{x_i^j \mid j \neq i\}$, terms A_i^j over V_j^p, and a well-founded domain $(A, \prec)$ with measure δ:

$$
\begin{array}{ll}
\text{PR1.} & M \ \vDash \ Env \rightarrow (p \Rightarrow (q \ \vee \ \bigvee_i \varphi_i^*)) \\
\text{PR2.} & M \ \vDash \ Env \ \rightarrow ((\varphi_i^* \ \wedge \ \delta = \alpha) \Rightarrow \varphi_i^* \, \mathcal{W} \, (q \ \vee \ \delta \prec \alpha)) \\
\text{PR3.} & M \ \vDash \ Env \ \rightarrow (\varphi_i^* \ \wedge \ \delta = \alpha \Rightarrow \Diamond(q \ \vee \ \delta \prec \alpha \ \vee \ \psi_i^*)) \\
\text{PR4.} & M_i \ \vDash \ Env_i \ \rightarrow ((\varphi_i \ \wedge \ \delta_i = \alpha_i \ \wedge \ \Box\Diamond \psi_i) \Rightarrow \Diamond(q \ \vee \ \delta_i \prec \alpha_i)) \\
\text{PR5.} & M_i \ \vDash \ Env_i \ \rightarrow (taken(M_i) \Rightarrow \bigwedge_{j \neq i} \mathcal{E}_j[A_j^i / x_j^i]) \\
\text{PR6.} & \vDash (\mathcal{E} \ \wedge \ x_j^{i\prime} = x_j^i) \ \rightarrow \ \bigwedge_i \mathcal{E}_i \\
\hline
 & M \ \vDash \ Env \ \rightarrow (p \Rightarrow \Diamond q)
\end{array}
$$

Proof:

Rule `PAR-CRESP` is relatively complete for response properties. According to the global proof rule **F-RESP**, presented and proved to be relatively complete in [MP91], there are past formulas χ_i and helpful transitions τ_i for $i = 1, \ldots, K$ such that a p-state begins a sequence of χ-states until a q-state, where $\chi = \bigvee_i \chi_i$, and every χ_i-state persists until τ_i is taken, which results in either q or a decrease in the well-founded measure. Then φ_i and ψ_i, for $i = 1, \ldots, M$, may be chosen to be the disjunction of χ_j and the disjunction of $\chi_j \ \wedge \ En(\tau_j)$, respectively, for each τ_j attributed to component i. ∎

Example: The response property

$$Env \ \rightarrow (at\_m_2[i] \Rightarrow \Diamond \ at\_m_3[i])$$

is valid for the resource allocator of Figure 3.5, where Env states that neither α nor β are modified by the environment. Apply rule PAR-CRESP, taking:

$$\begin{aligned} \mathcal{E}_i &: \ true \\ \varphi_i &: \ at\_m_2[i] \\ \psi_i &: \ |\alpha| = 0 \end{aligned}$$

$$\left.\begin{aligned} \mathcal{E}_j &: \ true \\ \varphi_j &: \ false \\ \psi_j &: \ false \end{aligned}\right\} \text{for } j = A, \text{ or } j \neq i$$

In particular, premise PR3

$$M \ \models \ Env \ \rightarrow (at\_m_2[i] \Rightarrow \Diamond(at\_m_3[i] \ \vee (|\alpha| = 0)))$$

follows from (3.3), established earlier. ■

3.5 Synchronous Communication

Message passing via synchronous channels typically requires coordination between a process and its environment. For instance, when one process sends a message, the environment must simultaneously receive the message. Thus, each communication event along a synchronous channel is both a system step and an environment step. Consequently, a number of subtle changes accompany the introduction of message passing through synchronous channels.

A synchronous channel may be declared as follows:

α : **channel**

Each synchronous channel may be shared by at most two processes. No new communication statements are introduced, but the *communication transitions* associated with the statements

ℓ_s: $\alpha \Leftarrow e$; $\widehat{\ell}_s$:
ℓ_r: $\alpha \Rightarrow x$; $\widehat{\ell}_r$:

for the case that α is a synchronous channel are given as follows:

$$\tau_{\ell_s} : \; moves(\pi, \ell_s, \widehat{\ell_s}) \;\wedge\; ready[\alpha \not\rightarrow] \;\wedge\; \textstyle\bigwedge_{u \in V - \{\pi\}} (u' = u)$$
$$\tau_{\ell_r} : \; moves(\pi, \ell_r, \widehat{\ell_r}) \;\wedge\; ready[\alpha \not\leftarrow] \;\wedge\; \textstyle\bigwedge_{u \in V - \{\pi, x\}} (u' = u)$$

where $ready[\alpha \not\rightarrow]$ and $ready[\alpha \not\leftarrow]$ are new, boolean auxiliary variables intended to signify whether the environment is ready to receive or ready to send, respectively. In these communication transitions, the variables which remain unchanged by each transition are explicitly stated. In particular, the conjunct $(x' = x)$ is not present in τ_{ℓ_r}, so x may be assigned any value when τ_{ℓ_r} is taken. Also, each communication transition may arbitrarily modify variables that are not contained in V, i.e., variables belonging to the environment. Thus, a state s' is a τ-*communication successor* of s, where τ is a communication transition, if:

$$\langle s, s' \rangle \;\models\; \tau$$

In contrast, recall that a state s' is a τ-*successor* of s, where τ is not a communication transition, if:

$$\langle s, s' \rangle \;\models\; \tau \;\wedge\; \bigwedge_{u \notin V} (u' = u)$$

3.5.1 Parallel Composition

Synchronous communication transitions are handled separately when constructing the parallel composition of two transition modules.

For each pair of *matching communication transitions*, i.e., τ_{ℓ_s} in M_1 and τ_{ℓ_r} in M_2 representing the following statements

$$\ell_s\!: \; \alpha \Leftarrow e; \quad \widehat{\ell_s}\!:$$
$$\ell_r\!: \; \alpha \Rightarrow x; \quad \widehat{\ell_r}\!:$$

respectively, the *joint communication transition* τ is defined to be

$$\tau : \; moves(\pi_1, \ell_s, \widehat{\ell_s}) \;\wedge\; moves(\pi_2, \ell_r, \widehat{\ell_r}) \;\wedge\; x' = e \;\wedge \bigwedge_{u \in V - \{\pi_1, \pi_2, x\}} (u' = u)$$

where a state s' is a *τ-successor* of s if:

$$\langle s, s' \rangle \;\vDash\; \tau \;\wedge\; \bigwedge_{u \notin V} (u' = u)$$

In other words, the joint transition τ is treated as an ordinary transition in the composed system, except for the associated compassion requirement. Furthermore, the channel variable α is included in the private variables of the composed system.

Each unmatched communication transition in M_1 is then modified by adding conjuncts to disallow unintended modification of variables of M_2, and vice versa. For instance, the following unmatched communication transition of M_1

$$moves(\pi_1, \ell_s, \widehat{\ell}_s) \;\wedge\; ready[\alpha \rightarrowtail] \;\wedge \bigwedge_{u \in V_1 - \{\pi\}} (u' = u)$$

would yield the communication transition

$$moves(\pi_1, \ell_s, \widehat{\ell}_s) \;\wedge\; ready[\alpha \rightarrowtail] \;\wedge \bigwedge_{u \in V_1 - \{\pi\}} (u' = u) \;\wedge\; \bigwedge_{u \in V_2} (u' = u)$$

in the combined transition module.

It is not difficult to establish the following, weaker version of Theorem 3.1.1.

Theorem 3.5.1 $Comp(M) \subseteq Comp(M_1) \cap Comp(M_2)$

3.5.2 Verification

For the case of modules communicating through synchronous channels, the environment restriction may refer to the private variables of the modules. This was not necessary previously, for the shared variables case or the asynchronous communication case, because the environment transition could not modify private variables.

Furthermore, when constructing the embedded transition module $[M, Env]$, each communication transition τ in M is replaced by the transition $\tau \wedge \mathcal{E}$ in $[M, Env]$.

Finally, assume that M_i and M_j share some channel α. Just as each auxiliary variable x_i^j in the specification of φ_i must be replaced by some A_i^j over V_j^p, there must be some $A_i^{j,s}$ and $A_i^{j,r}$ over V_j to replace $ready[\alpha \leftarrowtail]$ and $ready[\alpha \rightarrowtail]$ in M_i, and vice versa.

$$\begin{array}{ll} \alpha, \beta & : \textbf{channel} \\ u & : \textbf{integer} \end{array}$$

$$M_1 :: \begin{bmatrix} \ell_0\text{:} & \alpha \Leftarrow 0 \\ \ell_1\text{:} & \beta \Rightarrow u \end{bmatrix} \quad || \quad M_2 :: \begin{bmatrix} m_0\text{:} & \alpha \Rightarrow u \\ m_1\text{:} & \beta \Leftarrow 1 \end{bmatrix}$$

Figure 3.6: Program `SYNC-PING-PONG`.

Example: The following property holds for Program `SYNC-PING-PONG`:

$$Env \ \rightarrow (\Theta \Rightarrow \Diamond\, u = 1)$$

First, observe that

$$M_1 \ \models \ Env_1 \ \rightarrow \ (at\_\ell_0 \ \wedge \ ready[\alpha \nrightarrow]) \\ \Rightarrow \begin{pmatrix} at\_\ell_0 \ \wedge \ ready[\alpha \nrightarrow] \\ \vee \\ at\_\ell_1 \ \wedge \ ready[\beta \nleftarrow] \end{pmatrix} \mathcal{W}\,(u = 1)$$

may be established by choosing:

$$\mathcal{E}_1 : \quad (at\_\ell_0 \ \wedge \ ready[\alpha \nrightarrow]) \ \rightarrow \begin{pmatrix} at\_\ell_0{}' \ \wedge \ ready[\alpha \nrightarrow]' \\ \vee \\ at\_\ell_1{}' \ \wedge \ ready[\beta \nleftarrow]' \end{pmatrix}$$

$$(at\_\ell_1 \ \wedge \ ready[\beta \nleftarrow]) \ \rightarrow (at\_\ell_1{}' \ \wedge \ ready[\beta \nleftarrow]') \ \vee (u' = 1)$$

Taking $ready[\alpha \nrightarrow]$ and $ready[\beta \nleftarrow]$ to be at_m_0 and at_m_1, respectively, establishes

$$M \ \models \ Env \ \rightarrow (at\_\ell_0 \ \wedge \ at\_m_0) \Rightarrow \begin{pmatrix} at\_\ell_0 \ \wedge \ at\_m_0 \\ \vee \\ at\_\ell_1 \ \wedge \ at\_m_1 \end{pmatrix} \mathcal{W}\,(u = 1)$$

which is the first premise of rule `PAR-RESP`, where φ_1 and φ_2 are:

$$\varphi_1 : \quad (at\_\ell_0 \ \wedge \ ready[\alpha \nrightarrow]) \ \vee (at\_\ell_1 \ \wedge \ ready[\beta \nleftarrow])$$

$$\varphi_2 : \quad \mathit{false}$$

It should not be surprising to discover that the proof can be completed by assigning all responsibility for progress, i.e., φ_1, to M_1. There are really only two transitions in this system, and each can be attributed to either M_1 or M_2. ▪

3.6 Related Work

The results presented in this chapter derive most directly from the reactive modules of [MP92]. The reactive modules of [MP92] are not fully compositional, however, in the sense that the parallel composition of reactive modules yields a transition system. Thus, the result of composing two reactive modules cannot again be composed, and conversely, it is not possible to repeatedly reduce the verification task of a large system to the verification of smaller and smaller components. Furthermore, [MP92] considers only the very strong notion of *modular validity*, wherein a specification is modularly valid only if it holds for every computation of the module in an arbitrary environment. This leads to the strategy of *layered decomposition*, which has been reconsidered in the more general framework of this thesis.

There have been a number of early attempts to construct Hoare-style proof systems for parallel programs. [OG76a] proposes an *interference freedom test* to ensure that the actions of one system component do not invalidate the proof of another component. The interference freedom test, however, is noncompositional, since the global property of interference freedom can only be verified by considering the atomic actions of each component, rather than by establishing properties of each component.

[Jon83] introduces the *rely-guarantee* paradigm for *modular specification*. The environment restrictions introduced earlier are essentially *rely conditions*, i.e., restrictions on the transition relation of the environment, tailored for each proof. A complete proof system based solely on rely-guarantee conditions, however, is too cumbersome to be practical.

In the Unity framework [CM88], compositional verification is based on the Union Theorem and the introduction of *conditional properties*. The Union Theorem applies to safety properties and to simple liveness properties, but conditional properties must be used to verify more general liveness properties. A conditional property consists of a hypothesis and a conclusion, each of which may contain properties of a module, the module's environment, or the module and environment taken together. This verification style is very similar to the layered decomposition strategy.

Several compositional proof systems for processes communicating through synchronous channels are developed extensively in [ZdR89] and [Zwi89]. These systems are based on communication histories and may be applied to safety properties. [PJ91] considers liveness properties in a similar framework.

[AL93] draws a distinction between *composition* and *decomposition* of program specifications, roughly corresponding to the notions of *modularity* and *compositionality* in [ZdR89]. [AL93] considers primarily the case in which each variable is modified by at most one process, briefly describing the modifications required for the more general case where several processes modify a shared variable. In [AL93] the environment assumption of each process is verified by checking that it is implied by the higher-level specification, instead of checking each of the other component processes.

Chapter 4

Sequential Composition and Iteration

This chapter considers two possible semantics for sequential composition and, by extension, iteration.

For the first case, the computations of a transition module remain "anchored" to the first state of a state sequence, i.e., the first state s_0 must be initial, so this is referred to as *anchored composition*. For each computation σ of the sequential composition $M_1; M_2$, there are computations σ_1 and σ_2 of M_1 and M_2 such that $\sigma = \sigma_1 \star \sigma_2$. Note that $\star$ denotes the *fusion* of state sequences σ_1 and σ_2, to be

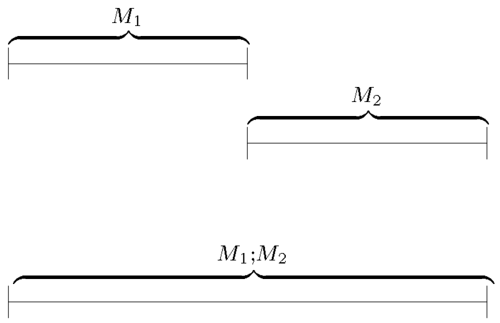

Figure 4.1: Anchored sequential composition.

defined later. A new temporal operator *follows*, denoted $\mathcal{F}$, is introduced to express the interaction between σ_1 and σ_2.

For the next case, the computations of a transition module are redefined to be "floating," i.e., some state s_i is initial, and the prefix $s_0, \ldots, s_{i-1}$ represents the history preceding the actual computation of the module. Thus, a computation of $M_1; M_2$ is

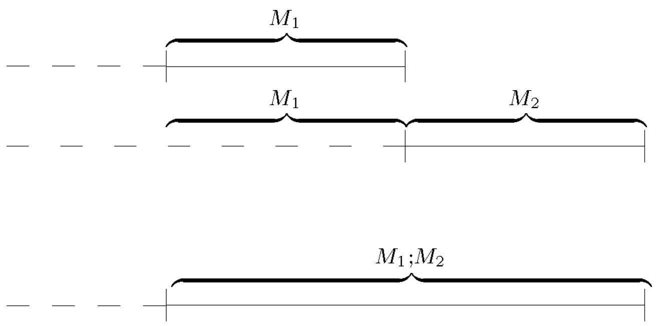

Figure 4.2: Floating sequential composition.

typically also a computation of M_2, such that a computation of M_1 appears before the computation of M_2. Unlike the anchored case, no new temporal operator is required. However, a floating semantics for temporal logic is used.

A number of other changes accompany sequential composition. Each transition module is parameterized by a boolean control predicate, used to define composition operations on transition modules. The initial condition Θ of a transition module is restricted to V^p, the private variables of the module. The initial condition for the shared variables is to be stated explicitly as part of the specification. Each transition module also includes an associated *termination condition*, i.e., an assertion over V^p, characterizing states in which the module has properly terminated. Furthermore, computations of a transition module may be either finite or infinite, and the last state of a finite computation is required to satisfy the termination condition.

4.1 Transition Modules

In the informal example of Figure 4.3, program module P_1 may be modeled by a

$$P_1 :: \begin{bmatrix} \ell_0\colon & x := x+1 \\ \ell_1\colon & \end{bmatrix} \quad || \quad P_2 :: \begin{bmatrix} m_0\colon & y := y+1 \\ m_1\colon & \end{bmatrix}$$

Figure 4.3: An informal example.

transition module $M_1(\zeta_1)$, where ζ_1 is the boolean control parameter. $M_1(\zeta_1)$ consists of the single transition τ_1

$$\tau_1 : \quad \zeta_1 \wedge at\_\ell_0 \wedge (at\_\ell_1)' \wedge (x' = x+1)$$

stating that when ζ_1 holds, P_1 may move from a state in which control is at ℓ_0 to a state in which control is at ℓ_1, while the value of x is incremented. Program module P_2 may be modeled similarly, by a transition module $M_2(\zeta_2)$ with the sole transition τ_2:

$$\tau_2 : \quad \zeta_2 \wedge at\_m_0 \wedge (at\_m_1)' \wedge (y' = y+1)$$

The sequential composition of $M_1(\zeta_1)$ and $M_2(\zeta_2)$ may then be taken to be the transition module $M(\zeta)$, whose transitions are taken to be the transitions of $M_1(\zeta)$ and $M_2(\zeta \wedge at\_\ell_1)$:

$$\begin{aligned} \widehat{\tau}_1 : &\quad \zeta \wedge at\_\ell_0 \wedge (at\_\ell_1)' \wedge (x' = x+1) \\ \widehat{\tau}_2 : &\quad \zeta \wedge at\_\ell_1 \wedge at\_m_0 \wedge (at\_m_1)' \wedge (y' = y+1) \end{aligned}$$

In this case, the initial condition of the transition module $M(\zeta)$ is the conjunction of the initial conditions of its components, i.e., $at\_\ell_0 \wedge at\_m_0$.

A *transition module* $M(\zeta)$ consists of the following components:

- V: A finite set of variables, partitioned into private variables V^p and shared variables V^s. The boolean control parameter ζ does not belong to V.

- $\mathcal{T}(\zeta)$: A finite set of *transitions*, i.e., assertions over V, V', and ζ. A state s' is a τ-*successor* of s if

$$\langle s, s' \rangle \vDash \tau \wedge \bigwedge_{u \notin V} (u' = u)$$

For every τ-successor s' of a state s, where $\tau \in \mathcal{T}(\zeta)$, there must be some $u \in V^p$ such that $s'[u] \neq s[u]$.

- Θ^p: An *initial condition*, i.e., an assertion over V^p characterizing the initial states of the transition module.
- Ω^p: A *termination condition*, i.e., an assertion over V^p characterizing the terminal states of the module. No state may be both initial and terminal, and every transition must be disabled on a terminal state.

The transition module $M(\zeta)$ may be referred to simply as M when omitting the parameter ζ creates no ambiguity. Given a transition module M with transitions $\mathcal{T}$, the set of transitions $\mathcal{T}(p)$ is obtained from $\mathcal{T}$ by replacing ζ by p in each transition of $\mathcal{T}$, for any assertion p. No transition in $\mathcal{T}(\mathit{false})$ may be enabled in any state.

4.1.1 The Environment Transition

As before, the environment transition τ_E of M is defined such that a state s' is a τ_E-successor of s if

$$\langle s, s' \rangle \vDash \bigwedge_{u \in V^p} (u' = u)$$

i.e., the environment may not modify any private variable of M.

4.1.2 Composing Transition Modules

Previously, the parallel composition of two transition modules was obtained by taking the union of the sets of transitions of each component, so that the set of computations of the composed system consisted of exactly the intersection of the sets of computations of each component. Other forms of composition may be defined by introducing *frames*, which are transition modules governing more complex interaction among constituent transition modules. Specifically, transition modules $M_1, \ldots, M_N$ may be composed in the frame M_f if the following conditions hold:

- The variables V_i and private variables V_j^p of transition modules M_i and M_j are disjoint for each $i \neq j$.
- The private variables V_i^p of each transition module M_i are also private variables of the frame, i.e., $V_i^p \subseteq V_f^p$, for each $i = 1, \ldots, N$.

Parallel Composition

For example, given transition modules M_1 and M_2 and the following frame:

- $V_f = V_1 \cup V_2$

$$V_f^p = V_1^p \cup V_2^p$$
$$V_f^s = V_1^s \cup V_2^s$$

- $\mathcal{T}_f(\zeta_f) = \emptyset$
- $\Theta_f^p = \Theta_1^p \ \wedge \ \Theta_2^p$
- $\Omega_f^p = \Omega_1^p \ \wedge \ \Omega_2^p$

it is trivial to define the parallel composition of M_1 and M_2:

- $V = V_f$

$$V^p = V_f^p$$
$$V^s = V_f^s$$

- $\mathcal{T}(\zeta) = \mathcal{T}_1(\zeta) \cup \mathcal{T}_2(\zeta)$
- $\Theta^p = \Theta_f^p$
- $\Omega^p = \Omega_f^p$

Sequential Composition

The sequential composition of M_1 and M_2 can be defined in the same frame:

- $V = V_f$

$$\begin{aligned} V^p &= V_f^p \\ V^s &= V_f^s \end{aligned}$$

- $\mathcal{T}(\zeta) = \mathcal{T}_1(\zeta) \cup \mathcal{T}_2(\zeta \ \wedge \ \Omega_1^p)$
- $\Theta^p = \Theta_f^p$
- $\Omega^p = \Omega_f^p$

Note that the transitions of M_2 may be enabled only when Ω_1^p holds, i.e., after M_1 has terminated.

Iteration

Given transition module M_1 and condition c, the frame M_f^c is defined:

- $V_f = V_1 \cup \{\pi\}$

$$\begin{aligned} V_f^p &= V_1^p \cup \{\pi\} \\ V_f^s &= V_1^s \end{aligned}$$

- $\mathcal{T}_f(\zeta_f) = \{\tau\}$

$$\tau : \ \zeta_f \ \wedge (\pi = 0) \ \wedge \ \Omega_1^p \ \wedge ((c \ \wedge \ \Omega_1^{p\prime}) \ \vee (\neg c \ \wedge \ \pi' = 1))$$

- $\Theta_f^p = \Omega_1^p \ \wedge (\pi = 0)$
- $\Omega_f^p = (\pi = 1)$

Then the iteration of M_1 with respect to the condition c is given by:

- $V = V_f$

$$V^p = V_f^p$$
$$V^s = V_f^s$$

- $\mathcal{T}(\zeta) = \mathcal{T}_f(\zeta) \cup \mathcal{T}_1(\zeta)$

- $\Theta^p = \Theta_f^p$

- $\Omega^p = \Omega_f^p$

4.2 Anchored Composition

4.2.1 Computations of a Transition Module

The length of a finite sequence of states $\sigma : s_0, \ldots, s_n$ is defined $|\sigma| = n$. The length of an infinite sequence of states σ is defined $|\sigma| = \infty$. Let $\mathcal{V}$ be a *vocabulary* containing V. A finite or infinite sequence of $\mathcal{V}$-states $\sigma : s_0, s_1, \ldots$ is defined to be a computation of M if it satisfies the following requirements:

- *Initiation* The state s_0 is an initial state.
- *Consecution* For every j, where $0 \leq j < |\sigma|$:
  - either the state s_{j+1} is a τ-successor of the state s_j for some $\tau \in \mathcal{T}(\mathit{true})$, i.e., transition τ was *taken* at position j in σ,
  - or the state s_{j+1} is a τ_E-successor of state s_j, i.e., an *environment step* is taken at position j in σ.
- *Termination* If σ is finite, then $s_{|\sigma|}$ is a terminal state.
- *Justice* If σ is infinite, then for each $\tau \in \mathcal{T}(\mathit{true})$ it is not the case that τ is continually enabled beyond some point in σ but taken at only finitely many positions in σ.

The set of computations of M is denoted $\mathit{Comp}(M)$. A computation containing a terminal state is a *terminating computation*. Note that terminating computations may be infinite, since a terminating computation may have an infinite suffix of terminal states.

Let $\star$ be the fusion[1] operator.

Theorem 4.2.1 $Comp(M_1; M_2) \subseteq Comp(M_1) \star Comp(M_2)$

Proof:

Let σ be a computation of $M = M_1; M_2$. If there is no position j such that s_j satisfies Ω_1^p, then σ must be infinite. Observe that no transition in $\mathcal{T}_2(\Omega_1^p)$ is enabled when Ω_1^p is false. It follows by induction that every transition taken in the computation σ of M is either from M_1 or is the environment transition. Every environment step of M is also an environment step of M_1, so it follows that σ is an infinite computation of M_1.

Otherwise, let s_j be the first Ω_1^p-state. It follows, from the argument given above, that $\sigma[0\ldots j]$ is a computation of M_1. Since Θ_2^p holds initially and neither M_1 nor the environment of M may change Θ_2^p to false, s_j is also a Θ_2^p-state. Similarly, Ω_1^p must continue to hold at every position beyond j, so every transition taken from j onward in the computation σ of M is either from M_2 or is the environment transition. Every environment step of M is also an environment step of M_2, hence $\sigma[j\ldots]$ is a computation of M_2. ▟

Notice that the converse to the above theorem does not hold. In particular, computations of M_1 and M_2 allow environment steps which modify private variables of M_2 and M_1, respectively. Furthermore, terminating computations of M_1 may be infinite, and may therefore belong to $Comp(M_1) \star Comp(M_2)$ without necessarily being computations of M.

[1]The *fusion* of sequences σ and σ', denoted $\sigma \star \sigma'$, is defined to be σ if σ is infinite. If $\sigma : s_0, s_1, \ldots, s_k$ is finite and the first state s'_0 of σ' equals s_k, then $\sigma \star \sigma'$ is given by $s_0, s_1, \ldots, s_k(= s'_0), s'_1, \ldots$. Otherwise $\sigma \star \sigma'$ does not exist.

The fusion of sets of state sequences $\Pi \star \Pi'$ includes all sequences $\sigma \star \sigma'$, where $\sigma \in \Pi$ and $\sigma' \in \Pi'$.

4.2.2 Temporal Logic with Follows

The subsequence $\sigma[j \ldots k]$ of a state sequence $\sigma = s_0, s_1, \ldots$ is given by $s_j, \ldots, s_k$. The j-th suffix of σ, denoted $\sigma[j \ldots]$, is given by $s_j, s_{j+1}, \ldots$.

Let $\mathcal{F}$ be the *follows* operator, where

$$\begin{aligned}(\sigma, j) \vDash p\,\mathcal{F}\,q \iff & \text{ for some } i,\ 0 \leq i \leq j,\ (\sigma, i) \vDash q \\ & \text{ and } (\sigma[i \ldots], j-i) \vDash p\end{aligned}$$

The language obtained by adding $\mathcal{F}$ to temporal logic is called TL$\mathcal{F}$.

If p and q are past formulas, then $(\sigma, j) \vDash p\,\mathcal{F}\,q$ depends only on the prefix $\sigma[0 \ldots j]$. Figure 4.4 illustrates that, if $(\sigma_1, i) \vDash q$ and $(\sigma_2, j-i) \vDash p$, then $(\sigma_1 \star \sigma_2, j) \vDash p\,\mathcal{F}\,q$.

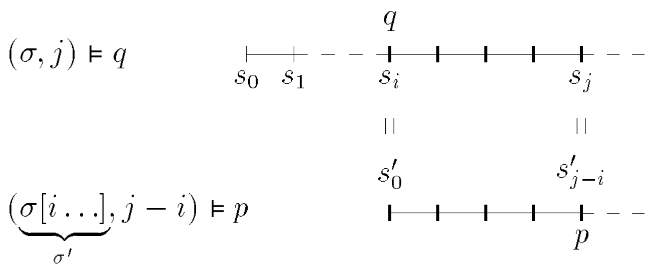

Figure 4.4: Showing that $(\sigma, j) \vDash\ p\,\mathcal{F}\,q$.

Example: Given:

$$\begin{aligned}p:&\quad z = 1 \ \rightarrow\ \diamondminus(x = 1)\ \vee\ \boxminus(y = 1)\\ q:&\quad y = 1 \ \rightarrow\ \diamondminus(x = 1)\\ r:&\quad z = 1 \ \rightarrow\ \diamondminus(x = 1)\end{aligned}$$

Observe that:

$$\vDash\ (p\,\mathcal{F}\,q) \Rightarrow r$$

Consider an arbitrary state sequence σ, and assume that $p\,\mathcal{F}\,q$ holds at position j. If $z = 1$ does not hold at j, then r is trivially true at j. Otherwise, choose i such that $(\sigma, i) \vDash q$ and $(\sigma[i, \ldots], j-i) \vDash p$. Then $z = 1$ holds at position $j-i$ of

$\sigma[i, \ldots]$, so by p either $x = 1$ at some position k, for $i \le k \le j$, or $y = 1$ holds at every k, for $i \le k \le j$. In the first case, $\diamondminus x = 1$ holds at j in σ, establishing r at j in σ. In the second case, $y = 1$ holds at i, so by q, there is some position k, for $0 \le k \le i$, such that $x = 1$ holds at k, and consequently $\diamondminus x = 1$ holds at j in σ, establishing r at j in σ. ∎

The *requires* operator $\mathcal{R}$ may be defined as follows:

$$p \,\mathcal{R}\, q \iff \neg(p \,\mathcal{F}\, \neg q)$$

Thus, $(\sigma_1 \star \sigma_2, j) \vDash p \,\mathcal{R}\, q$ and $(\sigma_2, j - |\sigma_1|) \vDash p$ implies $(\sigma_1, |\sigma_1|) \vDash q$.

A Decision Procedure for TL$\mathcal{F}$

Proposition 4.2.2 *TL$\mathcal{F}$ has a non-elementary decision procedure.*

Proof:

Let $SF(\Sigma)$ be the set of star-free expressions over alphabet Σ. Define a mapping $\varphi : SF(\Sigma) \mapsto \mathrm{TL}\mathcal{F}$ as follows:

- $\varphi_p = \circleddash(p \wedge \mathit{first})$ for $p \in \Sigma$
- $\varphi_{\neg p} = \neg \varphi_p$
- $\varphi_{p \cdot q} = \varphi_q \,\mathcal{F}\, \varphi_p$

Now, by structural induction, establish that for any $\alpha \in SF(\Sigma)$:

$$\sigma \in \mathcal{L}(\alpha) \quad \leftrightarrow \quad (\sigma \cdot \Sigma, |\sigma| + 1) \vDash \varphi_\alpha$$

Only the case $\alpha = p \cdot q$ is nontrivial. Let $\sigma \in \mathcal{L}(\alpha)$ be given as follows:

$$\sigma : \quad \overbrace{s_0, s_1, \ldots, s_j}^{\sigma_p \in \mathcal{L}(p)}, \overbrace{s_{j+1}, s_{j+2}, \ldots, s_k}^{\sigma_q \in \mathcal{L}(q)}$$

Then by the induction hypothesis:

$$(\sigma_q \cdot \Sigma, k - j) \vDash \varphi_q$$
$$(\sigma_p \cdot s_{j+1}, j + 1) \vDash \varphi_p$$

which establishes $(\sigma \cdot \Sigma, |\sigma| + 1) \models \varphi_q \,\mathcal{F}\, \varphi_p$. The converse is similar. [Sto74] established that the emptiness problem for star-free sets is non-elementary, so it follows that satisfiability for TL$\mathcal{F}$ is non-elementary. ∎

The following decision procedure closely follows the decision procedure for PTL with the operator *chop* given in [RP86]. It is convenient to introduce the notion of floating satisfiability. A formula φ is *floating satisfiable* if there is some state sequence σ and position j such that $(\sigma, j) \models \varphi$.

Given a TL$\mathcal{F}$ formula φ, the *tableau* for φ is a finite graph $\mathcal{T}_\varphi = (W_\varphi, R_\varphi)$ with nodes W_φ, called *atoms*, and edges R_φ. The tableau is constructed by induction on the nesting depth of the follows operator $\mathcal{F}$ in φ, where $\mathcal{T}_p = (W_p, R_p)$ is assumed to be the previously constructed tableau for p, for each subformula $p\,\mathcal{F}\,q$ of φ.

Given a set of formulas C, let $\widehat{C}$ denote the conjunction of all formulas in C.

The *closure* of a TL$\mathcal{F}$ formula φ is the smallest set of formulas $Cl(\varphi)$ containing φ and satisfying:

$$
\begin{array}{lll}
\bigcirc\, true, \ominus\, true \in Cl(\varphi) & & \\
p \in Cl(\varphi) & \Rightarrow & \neg p \in Cl(\varphi) \\
p \vee q \in Cl(\varphi) & \Rightarrow & p, q \in Cl(\varphi) \\
\bigcirc\, p \in Cl(\varphi) & \Rightarrow & p \in Cl(\varphi) \\
p\,\mathcal{U}\,q \in Cl(\varphi) & \Rightarrow & p, q, \bigcirc(p\,\mathcal{U}\,q) \in Cl(\varphi) \\
\ominus\, p \in Cl(\varphi) & \Rightarrow & p \in Cl(\varphi) \\
p\,\mathcal{S}\,q \in Cl(\varphi) & \Rightarrow & p, q, \ominus(p\,\mathcal{S}\,q) \in Cl(\varphi) \\
p\,\mathcal{F}\,q \in Cl(\varphi) & \Rightarrow & p, q, p\,\mathcal{F}\,true \in Cl(\varphi) \\
 & & \text{and } \bigcirc(\widehat{C}\,\mathcal{F}\,q), \ominus(\widehat{C}\,\mathcal{F}\,q) \in Cl(\varphi) \text{ for every } C \subseteq Cl(p)
\end{array}
$$

An atom for φ is a set of formulas $A \subset Cl(\varphi)$ such that:

$$
\begin{array}{lll}
true \in A & & \\
p \in A & \Longleftrightarrow & \neg p \notin A \\
p \vee q \in A & \Longleftrightarrow & p \in A \text{ or } q \in A \\
p\,\mathcal{U}\,q \in A & \Longleftrightarrow & q \in A \text{ or } p, \bigcirc(p\,\mathcal{U}\,q) \in A \\
p\,\mathcal{S}\,q \in A & \Longleftrightarrow & q \in A \text{ or } p, \ominus(p\,\mathcal{S}\,q) \in A
\end{array}
$$

$$
\begin{array}{lcl}
p\,\mathcal{F}\,q \in A & \Longleftrightarrow & \text{for some } C \in W_p\text{:} \\
 & & \quad \text{(a) } p \in C \\
 & & \quad \text{(b) } \widehat{C}\,\mathcal{F}\,q \in A \\
 & & \quad \text{(c) if } C \in \mathit{Init}(p) \text{ then } q \in A \\
 & & \quad \text{(d) if } C \notin \mathit{Init}(p) \text{ then } \ominus(\widehat{B}\,\mathcal{F}\,q) \in A \\
 & & \qquad\qquad \text{for some } (B, C) \in R_p \\
 & & \quad \text{(e) } \bigcirc(\widehat{D}\,\mathcal{F}\,q) \in A \text{ for some } (C, D) \in R_p
\end{array}
$$

where $\mathit{Init}(p)$ consists of the *initial atoms* of p, i.e., those atoms of p that contain the formula $\neg \ominus \mathit{true}$.

The set of atoms for φ is denoted $\mathit{Atoms}(\varphi)$.

The initial tableau $\mathcal{T}_{\varphi,0} = (W_{\varphi,0}, R_{\varphi,0})$ is a graph whose nodes are exactly the atoms of φ. There is an edge $(A_1, A_2) \in R_{\varphi,0}$ if the following conditions hold:

- $\bigcirc p \in A_1 \iff p \in A_2$
- $\ominus p \in A_2 \iff p \in A_1$

A path $\pi = A_0, A_1, A_2, \ldots$ in $\mathcal{T}_{\varphi,i}$ is *self-fulfilling* if the following conditions hold:

- A_0 is initial,
- for every $p\,\mathcal{U}\,q \in A_j$, there is some $k \geq j$ such that $q \in A_k$, and
- for every $p\,\mathcal{F}\,q \in A_j$, there is some path $B_k, \ldots, B_l$ in $\mathcal{T}_p$ such that $k \leq j \leq l$, $p \in B_j$, B_k is initial, $q \in A_k$, and for every i, $k \leq i \leq l$, $\widehat{B_i}\,\mathcal{F}\,q \in A_i$. In this case, $A_k, \ldots, A_l$ is said to *derive* $p\,\mathcal{F}\,q$ for A_j, and each $\widehat{B_i}\,\mathcal{F}\,q \in A_i$ is *derived from* $p\,\mathcal{F}\,q$ in A_j.

The path π *contains* φ if $\varphi \in A_i$ for some i. The path π *fulfills* φ if $\varphi \in A_0$.

Proposition 4.2.3

(a) The formula φ is satisfiable if and only if there is a path in $\mathcal{T}_{\varphi,0}$ that fulfills φ.

(b) The formula φ is floating satisfiable if and only if there is a self-fulfilling path in $\mathcal{T}_{\varphi,0}$ that contains φ.

Proof:

The proof proceeds by induction on the nesting depth of the follows operator $\mathcal{F}$ in φ. Let σ be a sequence that satisfies (or floating satisfies) φ. Take $\pi = A_0, A_1, \ldots$ to be the required path through $\mathcal{T}_{\varphi,0}$, where each A_i is an atom of φ defined as follows:

$$A_i : \quad \{p \in Cl(\varphi) \mid (\sigma, i) \vDash p\}$$

Conversely, for a path fulfilling φ (or a self-fulfilling path containing φ) $\pi = A_0, A_1, \ldots$, let σ be the obvious state sequence corresponding to π. It is straightforward to establish, by structural induction on the formulas in $Cl(\varphi)$, that $(\sigma, i) \vDash \widehat{A_i}$. ∎

A sequence of progressively smaller tableaus $\mathcal{T}_{\varphi,1}, \mathcal{T}_{\varphi,2}, \ldots$ may be obtained from $\mathcal{T}_{\varphi,0}$ by repeatedly removing atoms which cannot participate in paths fulfilling φ. In particular, $\mathcal{T}_{\varphi,i+1}$ is obtained from $\mathcal{T}_{\varphi,i}$ by identifying and removing an atom A in $\mathcal{T}_{\varphi,i}$ that satisfies one of the following conditions:

- A is not initial and has no incoming edges, or
- A has no outgoing edges, or
- for some $p\,\mathcal{U}\,q \in A$, there is no atom B reachable from A such that $q \in B$, or
- for some $p\,\mathcal{F}\,q \in A$, there is no path that derives $p\,\mathcal{F}\,q$ for A

Proposition 4.2.4 *Every self-fulfilling path in $\mathcal{T}_{\varphi,i}$ is preserved in $\mathcal{T}_{\varphi,i+1}$.*

Proof:

Obviously any $A \in \mathcal{T}_{\varphi,i} - \mathcal{T}_{\varphi,i+1}$ cannot belong to any self-fulfilling path. ∎

When $\mathcal{T}_{\varphi,k}$ is empty or cannot be reduced any further, the final tableau $\mathcal{T}_{\varphi}$ is taken to be $\mathcal{T}_{\varphi,k}$. It is not difficult to see that every atom in $\mathcal{T}_{\varphi}$ belongs to some self-fulfilling path. Consequently, the following proposition holds.

Proposition 4.2.5

(a) The formula φ is satisfiable if and only if there is an initial atom in $\mathcal{T}_\varphi$ containing φ.

(b) The formula φ is floating satisfiable if and only if there is an atom in $\mathcal{T}_\varphi$ containing φ.

A Deductive System

The deductive system below is an extension of the deductive system for PTL presented in [MP92].

F1. $\vdash \Box p \rightarrow p$

F2. $\vdash \bigcirc(p \rightarrow q) \Leftrightarrow (\bigcirc p \rightarrow \bigcirc q)$

F3. $\vdash \Box(p \rightarrow q) \Rightarrow (\Box p \rightarrow \Box q)$

F4. $\vdash \Box p \rightarrow \Box \bigcirc p$

F5. $(p \Rightarrow \bigcirc p) \rightarrow (p \Rightarrow \Box p)$

F6. $p \, \mathcal{W} \, q \Leftrightarrow (q \vee (p \wedge \bigcirc(p \, \mathcal{W} \, q)))$

F7. $\Box p \Rightarrow p \, \mathcal{W} \, q$

F8. $p \Rightarrow \bigcirc \ominus p$

N1. $(p \, \mathcal{F} \, q) \, \mathcal{F} \, r \Leftrightarrow p \, \mathcal{F} \, (q \, \mathcal{F} \, r)$

N2. $(p \vee q) \, \mathcal{F} \, r \Rightarrow (p \, \mathcal{F} \, r) \vee (q \, \mathcal{F} \, r)$

N3. $p \, \mathcal{F} \, (q \vee r) \Rightarrow (p \, \mathcal{F} \, q) \vee (p \, \mathcal{F} \, r)$

N4. $p \, \mathcal{F} \, q \Rightarrow p$ (for a proposition p)

N5. $\widetilde{\ominus} \mathit{false} \; \mathcal{F} \, q \Leftrightarrow q$

N6. $(\ominus p) \, \mathcal{F} \, q \Leftrightarrow \ominus(p \, \mathcal{F} \, q)$

N7. $(\bigcirc p) \, \mathcal{F} \, q \Leftrightarrow \bigcirc(p \, \mathcal{F} \, q)$

N8. $\mathit{true} \; \mathcal{F} \, q \Rightarrow \diamondminus q$

TGEN. $\Box p$ (for a valid state formula p)

P1. $\vdash \ominus p \Rightarrow \widetilde{\ominus} p$

P2. $\vdash \widetilde{\ominus}(p \rightarrow q) \Leftrightarrow (\widetilde{\ominus} p \rightarrow \widetilde{\ominus} q)$

P3. $\vdash \boxminus(p \rightarrow q) \Rightarrow (\boxminus p \rightarrow \boxminus q)$

P4. $\vdash \Box p \rightarrow \Box \widetilde{\ominus} p$

P5. $(p \Rightarrow \widetilde{\ominus} p) \rightarrow (p \Rightarrow \boxminus p)$

P6. $p \, \mathcal{B} \, q \Leftrightarrow (q \vee (p \wedge \widetilde{\ominus}(p \, \mathcal{B} \, q)))$

P7. $\widetilde{\ominus} \mathit{false}$

P8. $p \Rightarrow \widetilde{\ominus} \bigcirc p$

The deductive system uses the proof rules *modus ponens*

$$\mathrm{MP}: \quad p \rightarrow q,\, p \;\vdash\; q$$

and *instantiation*

$$\mathrm{INST}: \quad p \;\vdash\; p[\alpha]$$

where α is a replacement for the sentence symbols in p, as well as *composition monotonicity*

$$\mathrm{CM}: \quad p \Rightarrow p',\, q \Rightarrow q' \;\vdash\; p \,\mathcal{F}\, q \Rightarrow p' \,\mathcal{F}\, q'$$

and *impossibility*[2]:

$$\mathrm{IMP}: \quad \Box \neg p \;\vdash\; \Box \neg (p \,\mathcal{F}\, q)$$

The only remaining proof rule relies on a network of premises for its conclusion. Formally, an *index-table* T is a pair $(\overline{n}, \delta)$, where $\overline{n}$ is the set of indices $\{1, \ldots, n\}$ and $\delta \subseteq \overline{n} \times \overline{n}$ is the accessibility relation. The set of initial indices T^f is given by $\{j \in \overline{n} \mid (i,j) \notin \delta \text{ for every } i \in \overline{n}\}$. The closure of j, denoted T_j, is given by $\{i \in \overline{n} \mid (i,j) \in \delta^*\}$. Given formulas p_i and q_i for $i \in \overline{n}$, let $m \in \overline{n}$ and $j \in \mathrm{T}_m \cap \mathrm{T}^f$. The *graph induction* rule is given as follows:

$$
\begin{array}{llll}
\text{GIND:} & \text{(premises)} & \text{a.} & \text{for every } (k,l) \in \delta \\
 & & & \vdash q_l \Rightarrow ((p_l \;\wedge\; \widetilde{\ominus} p_k) \,\mathcal{F}\, \mathit{true} \rightarrow \widetilde{\ominus} q_k) \\
 & & \text{b.} & \text{for every } k \in \overline{n} \\
 & & & \vdash p_k \Rightarrow \widetilde{\ominus} \bigvee_{(j,k) \in \delta} p_j \\
 & & \text{c.} & \text{for every } k, l \in \overline{n}, k \neq l \\
 & & & \vdash \Box \neg (p_k \;\wedge\; p_l) \\
 & \text{(conclusion)} & \multicolumn{2}{l}{\vdash q_m \Rightarrow (p_m \;\wedge\; \diamondminus p_j) \,\mathcal{R}\, q_j}
\end{array}
$$

Proof:

The soundness of the graph induction rule may be established as follows. Let σ be a state sequence such that $(\sigma, k) \vDash q_m$, and for some $i \leq k$, $(\sigma[i, \ldots], k -$

[2]This rule is not listed in [RP86], but is necessary for completeness. For instance, $\neg(\neg\varphi \,\mathcal{F}\, q)$, where φ is a valid temporal formula, cannot be proven without it.

$i) \models p_m \wedge \diamondminus p_j$. It is necessary to show $(\sigma, i) \models q_j$. First observe $p_j \Rightarrow \widetilde{\circleddash} \mathit{false}$ by premise b, since $j \in \mathrm{T}^f$, so it must be the case that $(\sigma[i, \ldots], 0) \models p_j$.

Now, establish by induction that for each l from k down to i, there is some $n \in \mathrm{T}_m$ such that $(\sigma[i, \ldots], l - i) \models p_n$ and $(\sigma, l) \models q_n$. The base case, for $l = k$, is established by taking $n = m$. For the induction step, let n satisfy the desired property for l, where $i < l \leq k$. By premise b, there is some n' such that $(\sigma[i, \ldots], (l - 1) - i) \models p_{n'}$ and $(n', n) \in \delta$. It follows that $(\sigma, l) \models (p_n \wedge \circleddash p_{n'}) \mathcal{F} \mathit{true}$, and since $(\sigma, l) \models q_n$ by the induction hypothesis, premise a yields $(\sigma, l - 1) \models q_{n'}$.

Then, for $k = i$, there is some n such that $(\sigma[i, \ldots], 0) \models p_n$ and $(\sigma, i) \models q_n$. As observed earlier, $(\sigma[i, \ldots], 0) \models p_j$. By premise c, it must be the case that $j = n$, establishing $(\sigma, i) \models q_j$ as desired. ∎

If a theorem or derived rule of TL$\mathcal{F}$ is also listed as a theorem or derived rule of PTL in [MP92], then the proof is omitted below if it is identical to the corresponding proof in PTL. Applications of instantiation are not explicitly shown.

Rule TGI. $\Box p[\alpha]$ for a valid state formula p and a replacement α

Rule PAR. $\Box p \vdash p$

Rule EMP. $(p_1 \wedge \cdots \wedge p_n) \Rightarrow q, \Box p_1, \ldots, \Box p_n \vdash \Box q$

When the first premise of EMP is a substitution instance of an obvious propositional tautology, rule EMP may be applied without listing the first premise. This is referred to as an application of EPR, i.e., *entailment propositional reasoning.* Similarly, an application of PR (propositional reasoning) assumes that the first premise of *modus ponens* is understood.

Rule $\bigcirc$M. *a.* $p \Rightarrow q \vdash \bigcirc p \Rightarrow \bigcirc q$

b. $p \Leftrightarrow q \vdash \bigcirc p \Leftrightarrow \bigcirc q$

Theorem T1. $\bigcirc(p \wedge q) \Leftrightarrow (\bigcirc p \wedge \bigcirc q)$

Theorem T2. $\bigcirc(p \vee q) \Leftrightarrow (\bigcirc p \vee \bigcirc q)$

Rule CI. $p \Rightarrow \bigcirc p \vdash p \Rightarrow \Box p$

Theorem T3. $p \,\mathcal{U}\, q \Leftrightarrow (q \vee (p \wedge \bigcirc p \,\mathcal{U}\, q))$

Theorem T4. $p \,\mathcal{U}\, q \Rightarrow \Diamond q$

Theorem T5. $\Box p \;\Leftrightarrow\; p \wedge \bigcirc \Box p$

Rule $\Box$G. $\Box p \;\vdash\; \Box \Box p$

Rule $\Box$M. *a.* $p \Rightarrow q \;\vdash\; \Box p \Rightarrow \Box q$

b. $p \Leftrightarrow q \;\vdash\; \Box p \Leftrightarrow \Box q$

1.	$p \Leftrightarrow q$	premise
2.	$p \Rightarrow q$	1 EPR
3.	$\Box p \Rightarrow \Box q$	2 $\Box$M
4.	$q \Rightarrow p$	1 EPR
5.	$\Box q \Rightarrow \Box p$	4 $\Box$M
6.	$\Box p \Leftrightarrow \Box q$	3, 5 EPR

Theorem T6. $\Box \bigcirc p \;\Rightarrow\; \bigcirc \Box p$

1.	$\Box \bigcirc p \;\Leftrightarrow\; \bigcirc p \wedge \bigcirc \Box \bigcirc p$	T5
2.	$\bigcirc(p \wedge \Box \bigcirc p) \;\Leftrightarrow\; (\bigcirc p \wedge \bigcirc \Box \bigcirc p)$	T1
3.	$p \wedge \Box \bigcirc p \;\Rightarrow\; \bigcirc(p \wedge \Box \bigcirc p)$	1, 2 EPR
4.	$p \wedge \Box \bigcirc p \;\Rightarrow\; \Box(p \wedge \Box \bigcirc p)$	3 CI
5.	$p \wedge \Box \bigcirc p \;\Rightarrow\; p$	TGI
6.	$\Box(p \wedge \Box \bigcirc p) \;\Rightarrow\; \Box p$	$\Box$M
7.	$p \wedge \Box \bigcirc p \;\Rightarrow\; \Box p$	4, 6 EMP
8.	$\bigcirc(p \wedge \Box \bigcirc p) \;\Rightarrow\; \bigcirc \Box p$	7 $\bigcirc$M
9.	$\Box \bigcirc p \;\Rightarrow\; \bigcirc \Box p$	1, 2, 8 EPR

Rule $\bigcirc$F: *a.* $\bigcirc p \Rightarrow \bigcirc q \;\vdash\; \bigcirc(p \Rightarrow q)$

b. $\bigcirc p \Leftrightarrow \bigcirc q \;\vdash\; \bigcirc(p \Leftrightarrow q)$

1.	$\bigcirc p \Rightarrow \bigcirc q$	premise
2.	$\bigcirc(p \rightarrow q) \Leftrightarrow (\bigcirc p \rightarrow \bigcirc q)$	FX2
3.	$\Box \bigcirc(p \rightarrow q) \Rightarrow \bigcirc \Box(p \rightarrow q)$	T6
4.	$\bigcirc(p \Rightarrow q)$	1–3 EPR

1.	$(p \leftrightarrow q) \Leftrightarrow (p \rightarrow q \;\wedge\; q \rightarrow p)$	TGI

2.	$\bigcirc(p \leftrightarrow q) \Leftrightarrow \bigcirc(p \rightarrow q \;\;\wedge\;\; q \rightarrow p)$	$\bigcirc$M
3.	$\bigcirc(p \rightarrow q \;\;\wedge\;\; q \rightarrow p) \;\;\Leftrightarrow\;\; \bigcirc(p \rightarrow q) \wedge \bigcirc(q \rightarrow p)$	T1
4.	$\bigcirc(p \rightarrow q) \Leftrightarrow (\bigcirc p \rightarrow \bigcirc q)$	FX2
5.	$\bigcirc(q \rightarrow p) \Leftrightarrow (\bigcirc q \rightarrow \bigcirc p)$	FX2
6.	$(\bigcirc p \rightarrow \bigcirc q \;\;\wedge\;\; \bigcirc q \rightarrow \bigcirc p) \Leftrightarrow (\bigcirc p \leftrightarrow \bigcirc q)$	TGI
7.	$\bigcirc p \Leftrightarrow \bigcirc q$	premise
8.	$\Box \bigcirc(p \leftrightarrow q) \Rightarrow \bigcirc \Box(p \leftrightarrow q)$	T6
9.	$\bigcirc(p \Leftrightarrow q)$	2–8 EPR

Theorem T7. $p \;\Leftrightarrow\; \bigcirc \ominus p$

1.	$\neg p \Rightarrow \bigcirc \ominus \neg p$	FX8
2.	$\neg \bigcirc \ominus \neg p \Rightarrow p$	1 EPR
3.	$\neg \bigcirc \ominus \neg p \Leftrightarrow \bigcirc \widetilde{\ominus} p$	FX1
4.	$\bigcirc \widetilde{\ominus} p \Rightarrow \neg \bigcirc \ominus \neg p$	3 EPR
5.	$\bigcirc \widetilde{\ominus} p \Rightarrow p$	2, 4 EPR
6.	$\bigcirc(p \rightarrow q) \;\;\Leftrightarrow\;\; (\bigcirc p \rightarrow \bigcirc q)$	FX2
7.	$\Box \bigcirc(p \rightarrow q) \Leftrightarrow (\bigcirc p \Rightarrow \bigcirc q)$	6 $\Box$M
8.	$\Box \bigcirc(p \rightarrow q) \leftrightarrow (\bigcirc p \Rightarrow \bigcirc q)$	7 PAR
9.	$\ominus p \Rightarrow \widetilde{\ominus} p$	PX1
10.	$\Box p \rightarrow \Box \bigcirc p$	FX4
11.	$\Box \bigcirc(\ominus p \rightarrow \widetilde{\ominus} p)$	10, 9 MP
12.	$\bigcirc \ominus p \Rightarrow \bigcirc \widetilde{\ominus} p$	8, 11 PR
13.	$\bigcirc \ominus p \Rightarrow p$	5, 12 EPR
14.	$p \Rightarrow \bigcirc \ominus p$	FX8
15.	$p \Leftrightarrow \bigcirc \ominus p$	13, 14 EPR

Rule $\widetilde{\ominus}$G: $\Box p \;\vdash\; \Box \widetilde{\ominus} p$

Rule $\widetilde{\ominus}$M: $p \Rightarrow q \;\vdash\; \widetilde{\ominus} p \Rightarrow \widetilde{\ominus} q$

Theorem T8. $\ominus(p \wedge q) \;\Leftrightarrow\; (\ominus p \wedge \ominus q)$

Theorem T9. $\ominus(p \vee q) \;\Leftrightarrow\; (\ominus p \vee \ominus q)$

Theorem T10. $\widetilde{\ominus}(p \wedge q) \;\Leftrightarrow\; (\widetilde{\ominus} p \wedge \widetilde{\ominus} q)$

Theorem T11. $\widetilde{\ominus}(p \lor q) \Leftrightarrow (\widetilde{\ominus} p \lor \widetilde{\ominus} q)$

Rule $\ominus$M: $p \Rightarrow q \vdash \ominus p \Rightarrow \ominus q$

Theorem T12. $\ominus p \Leftrightarrow \widetilde{\ominus} p \land \ominus\, true$

Theorem T13. $\widetilde{\ominus} p \Leftrightarrow \ominus p \lor \widetilde{\ominus} false$

Completeness

Assume that φ is valid. The following proof proceeds by induction on the nesting depth of the follows operator $\mathcal{F}$ in φ. Construct the initial tableau $\mathcal{T}_{\neg\varphi,0} = (W_{\neg\varphi,0}, R_{\neg\varphi,0})$ for $\neg\varphi$ as described above.

Lemma 4.2.6 $\vdash \Box \bigvee_{A \in W_{\neg\varphi,0}} \widehat{A}$

Proof:

Take $\mathcal{A}$ to be

$$\mathcal{A}: \quad \{A \subset Cl(\varphi) \mid \forall p \in Cl(\varphi).\ p \in A \Longleftrightarrow \neg p \notin A\}$$

and observe $\vdash \Box \bigvee_{A \in \mathcal{A}} \widehat{A}$ by TGI. Since $Atoms(\varphi) \subseteq \mathcal{A}$, it suffices to show $\vdash \Box \neg \widehat{A}$ for any $A \in \mathcal{A} - Atoms(\varphi)$. The only interesting condition to consider in the definition of atoms for φ is the last one.

First, assume that $p \,\mathcal{F}\, q \in A$. Suppose there is no $C \in W_p$ such that $p \in C$, i.e., by proposition 4.2.5, $\models \Box \neg p$, and by induction, $\vdash \Box \neg p$. Rule IMP yields $\vdash \Box \neg (p \,\mathcal{F}\, q)$, which, when combined with $\vdash \widehat{A} \Rightarrow p \,\mathcal{F}\, q$, implies $\vdash \Box \neg \widehat{A}$. Otherwise, choose $C \in W_p$ such that $p \in C$ and $\widehat{C} \,\mathcal{F}\, q \in A$. Note that such a choice must be possible, since $\vdash p \Rightarrow \bigvee_{p \in C \in W_p} \widehat{C}$ follows from $\vdash \Box \bigvee_{C \in W_p} \widehat{C}$, yielding $\vdash p \,\mathcal{F}\, q \Rightarrow (\bigvee_{p \in C \in W_p} \widehat{C}) \,\mathcal{F}\, q$ by CM, and $\vdash \widehat{A} \Rightarrow p \,\mathcal{F}\, q$ holds by TGI. If $C \in Init(p)$, i.e., $\widetilde{\ominus} false \in C$, then $\vdash \widehat{C} \Rightarrow \widetilde{\ominus} false$ by TGI. Also, $\vdash \widehat{A} \Rightarrow \widehat{C} \,\mathcal{F}\, q$, so $\vdash \widehat{A} \Rightarrow \widetilde{\ominus} false \,\mathcal{F}\, q$ by CM. Then, by N5, $\vdash \widehat{A} \Rightarrow q$. If $C \notin Init(p)$, i.e., $\ominus\, true \in C$, apply Lemma 4.2.8 inductively to obtain $\vdash \widehat{C} \Rightarrow \widetilde{\ominus} \bigvee_{(B,C) \in R_p} \widehat{B}$. By T12 and CM, $\vdash \widehat{C} \,\mathcal{F}\, q \Rightarrow (\ominus \bigvee_{(B,C) \in R_p} \widehat{B}) \,\mathcal{F}\, q$, and by N6, N2 and T9, $\vdash \widehat{A} \Rightarrow \bigvee_{(B,C) \in R_p} \ominus(\widehat{B} \,\mathcal{F}\, q)$ as desired. Similarly, apply Lemma 4.2.8 inductively to obtain $\vdash \widehat{C} \Rightarrow \bigcirc \bigvee_{(C,D)\in R_p} \widehat{D}$. By CM, $\vdash \widehat{C}\,\mathcal{F}\,q \Rightarrow (\bigcirc \bigvee_{(C,D)\in R_p} \widehat{D})\,\mathcal{F}\,q$, By N7, N2 and T2, $\vdash \widehat{A} \Rightarrow \bigvee_{(C,D)\in R_p} \bigcirc(\widehat{D}\,\mathcal{F}\,q)$ as desired.

Conversely, assume that there is some $C \in W_p$ such that $p \in C$ and $\widehat{C}\,\mathcal{F}\,q \in A$, but $p\,\mathcal{F}\,q \notin A$. By TGI $\widehat{C} \Rightarrow p$, so by CM $\widehat{C}\,\mathcal{F}\,q \Rightarrow p\,\mathcal{F}\,q$. Hence $\vdash \widehat{A} \Rightarrow p\,\mathcal{F}\,q$, but since $\neg(p\,\mathcal{F}\,q)$ must be in A, i.e., $\vdash \widehat{A} \Rightarrow \neg(p\,\mathcal{F}\,q)$ by TGI, $\vdash \Box\neg\widehat{A}$ follows immediately. ∎

The following lemmas hold by induction for each tableau $\mathcal{T}_{\neg\varphi,0}, \mathcal{T}_{\neg\varphi,1}, \ldots$.

Lemma 4.2.7 $\vdash \Box\neg\widehat{A}$ *for each* $A \notin \mathcal{T}_{\neg\varphi,i}$

Proof:

The proof proceeds by induction. The base case is stated in Lemma 4.2.6. For the induction step, assume Lemmas 4.2.7 through 4.2.10 hold for some $i \geq 0$. Consider $A \in W_{\neg\varphi,i} - W_{\neg\varphi,i+1}$. If A is not initial, i.e., $\vdash \widehat{A} \Rightarrow \circleddash\, \mathit{true}$, and A has no incoming edges, i.e., $\vdash \widehat{A} \Rightarrow \widetilde{\circleddash} \mathit{false}$ by Lemma 4.2.8, then clearly $\vdash \Box\neg\widehat{A}$. It is similarly easy to show $\vdash \Box\neg\widehat{A}$ if A has no outgoing edges. If A contains the formula $p\,\mathcal{U}\,q$ but there is no atom B reachable from A that contains q, then $\vdash \widehat{A} \Rightarrow \Box\neg q$ follows from Lemma 4.2.9, and by T4, $\vdash \Box\neg\widehat{A}$. Finally, suppose A contains the formula $p\,\mathcal{F}\,q$ but there is no path that derives $p\,\mathcal{F}\,q$ for A. From Lemma 4.2.10 $\vdash \widehat{A} \Rightarrow p\,\mathcal{R}\,\mathit{false}$ holds, so by CM and the definition of $\mathcal{R}$, $\vdash \widehat{A} \Rightarrow \neg(p\,\mathcal{F}\,q)$. Since $p\,\mathcal{F}\,q \in A$, this implies $\vdash \Box\neg\widehat{A}$. ∎

Lemma 4.2.8 *For each* $A_2 \in W_{\neg\varphi,i}$*:*

$$\vdash \widehat{A_2} \Rightarrow \left[\left(\widetilde{\circleddash} \bigvee_{(A_1,A_2)\in R_{\neg\varphi,i}} \widehat{A_1}\right) \wedge \left(\widetilde{\bigcirc} \bigvee_{(A_2,A_3)\in R_{\neg\varphi,i}} \widehat{A_3}\right)\right]$$

Lemma 4.2.9 *For each* $A_2 \in W_{\neg\varphi,i}$*:*

$$\vdash \widehat{A_2} \Rightarrow \left[\left(\boxminus \bigvee_{(A_1,A_2)\in R_{\neg\varphi,i}} \widehat{A_1}\right) \wedge \left(\Box \bigvee_{(A_2,A_3)\in R_{\neg\varphi,i}} \widehat{A_3}\right)\right]$$

Lemma 4.2.10 *For each $A \in W_{\neg\varphi,i}$ such that $p\,\mathcal{F}\,q \in A$, observe that $p\,\mathcal{F}\,true \in A$ follows from CM and the definition of atoms for φ. For every atom $B \in W_p$, let ρ_B be the set of all $A_1 \in W_{\neg\varphi,i}$ such that $\widehat{B}\,\mathcal{F}\,true \in A_1$ is derived from $p\,\mathcal{F}\,true$ in A, and let ρ be the union of all ρ_B such that $B \in Init(p)$.*

$$\vdash \widehat{A} \Rightarrow p\,\mathcal{R}\,(\bigvee_{A_1 \in \rho} \widehat{A_1})$$

Proof:

Define:

$$U = \{D \in W_p \mid p \in D \text{ and } \widehat{D}\,\mathcal{F}\,q \in A\}$$
$$V = \{D \in Init(p) \mid (D, D') \in R_p^* \text{ for some } D' \in U\}$$

Note that U and V cannot be empty by the definition of atoms for φ.

Choose some enumeration of $W_p = \{D_1, D_2, \ldots, D_n\}$, and denote $q_j = \bigvee_{B \in \rho_{D_j}} \widehat{B}$ and $p_j = \widehat{D_j}$ for each $j = 1, \ldots, n$. Define the obvious index table over $\overline{n}$, taking $(i,j) \in \delta \iff (D_i, D_j) \in R_p$. Choose m and j such that D_m and D_j are in U and V respectively, and note that $j \in \mathrm{T}_m \cap \mathrm{T}^f$. To establish the first premise of the graph induction rule, consider some $(k,l) \in \delta$. If ρ_{D_l} is empty, then the premise is trivial. Otherwise, it is sufficient to establish

$$\vdash \widehat{B} \Rightarrow ((p_l \wedge \widetilde{\ominus} p_k)\,\mathcal{F}\,true \rightarrow \widetilde{\ominus} q_k) \tag{4.1}$$

for every $B \in \rho_{D_l}$. The second premise holds by Lemma 4.2.8, and the third premise holds by TGI, considering the definition of atoms for p. Consequently, by graph induction:

$$\vdash q_m \Rightarrow (\widehat{D_m} \wedge \diamondminus \widehat{D_j})\,\mathcal{R}\,q_j \tag{4.2}$$

Observe that Equation 4.2 holds by CM for the case $j \notin \mathrm{T}_m$; in particular, by Lemma 4.2.9, $\vdash \widehat{D_m} \Rightarrow \Box \neg \widehat{D_j}$. It follows that:

$$\vdash q_m \Rightarrow (\widehat{D_m} \wedge \diamondminus \bigvee_{D_j \in V} \widehat{D_j})\,\mathcal{R} \bigvee_{D_j \in V} q_j$$

Observe that $\bigvee_{D_j \in V} q_j$ is exactly $\bigvee_{A_2 \in \rho} \widehat{A_2}$, and by CM obtain:

$$\vdash q_m \Rightarrow \widehat{D_m}\, \mathcal{R} \bigvee_{A_2 \in \rho} \widehat{A_2} \tag{4.3}$$

Observe $\vdash \widehat{A} \Rightarrow \bigwedge D_m \in U p_m$, and since Equation 4.3 holds for every $D_m \in U$:

$$\vdash \widehat{A} \Rightarrow (\bigvee_{D_m \in U} \widehat{D_m})\, \mathcal{R} \bigvee_{A_2 \in \rho} \widehat{A_2}$$

Finally, observe $\vdash \widehat{A} \Rightarrow (p \rightarrow \bigvee_{D_m \in U} \widehat{D_m})$ and apply CM to obtain the desired result. ∎

Let $\mathcal{T}_{\neg\varphi,k}$ be the final tableau, denoted $\mathcal{T}_{\neg\varphi}$. Observe that $\vdash \Box \bigvee_{A \in W_{\neg\varphi}} \widehat{A}$ follows from Lemmas 4.2.6 and 4.2.7, so by rule PAR, $\vdash \bigvee_{A \in W_{\neg\varphi}} \widehat{A}$. Furthermore, $\vdash \neg\widehat{A}$ for any noninitial atom A, so $\vdash \bigvee_{A \in W^0} \widehat{A}$ for the set of initial atoms $W^0 \subseteq W_{\neg\varphi}$. Since φ is valid, $\neg\varphi$ is not satisfiable, and there cannot be any initial atoms in $\mathcal{T}_{\neg\varphi}$ containing $\neg\varphi$. Therefore φ is contained in every initial atom, yielding $\vdash (\bigvee_{A \in W^0} \widehat{A}) \rightarrow \varphi$, which establishes $\vdash \varphi$.

4.2.3 Verification

TL$\mathcal{F}$ does not have a simple axiomatization nor even an elementary decision procedure, which would seem to make it an unlikely language for verification. However, it is both possible and reasonable to restrict the use of the follows operator $\mathcal{F}$ in such a way that verification is straightforward and natural, as seen below. In particular, note that $\mathcal{F}$ need never appear in the specification of a transition module, and in fact appears only in premises of the form

$$\models (p\, \mathcal{F}\, q) \Rightarrow r$$

for past formulas p, q and r.

Note that, since the initial condition Θ^p of a transition module refers only to private variables V^p of the module, an initial condition Θ^s on the shared variables is included as part of the specification.

Proof Rules for Safety Properties

$$\begin{array}{|l|}\hline \texttt{SEQ}-\texttt{SAFE} \\ \text{For past formulas } p_1,\, p_2 \text{ and assertion } \Theta_2^s\text{:} \\ \quad \text{QS1.}\ \ M_1 \vDash (\Theta_1^s \,\wedge\, Env) \rightarrow \square\, p \\ \quad \text{QS2.}\ \ \vDash (p_2\, \mathcal{F}\, p_1) \Rightarrow p \\ \quad \text{QS3.}\ \ M_1 \vDash (\Theta_1^s \,\wedge\, Env) \rightarrow (\Omega_1^p \Rightarrow p_1 \,\wedge\, \Theta_2^s) \\ \quad \text{QS4.}\ \ M_2 \vDash (\Theta_2^s \,\wedge\, Env) \rightarrow \square\, p_2 \\ \hline \qquad\quad M \vDash (\Theta_1^s \,\wedge\, Env) \rightarrow \square\, p \\ \hline \end{array}$$

Proof:

A formula φ *holds at position* j *of* $\sigma_1 \star \sigma_2$ *after removing prefix* σ_1 if $j \geq |\sigma_1|$ and $(\sigma_2, j - |\sigma_1|) \vDash \varphi$.

Let σ be a computation of M satisfying Θ_1^s and Env. Let σ_1 and σ_2 be computations of M_1 and M_2 as described in the proof of Theorem 4.2.1. Consider an arbitrary position j. If $j \leq |\sigma_1|$, then j is a p-position by QS1. Otherwise, let $k = |\sigma_1|$. By QS3, p_1 and Θ_2^s hold at k. By QS4, p_2 holds at position j of $\sigma_1 \star \sigma_2$ after removing prefix σ_1, so $p_2\, \mathcal{F}\, p_1$ holds at position j of σ. By QS2, j must be a p-position. ∎

$$P_1 :: \begin{bmatrix} \ell_0\text{:}\ \ \textbf{if } x = 1 \textbf{ then} \\ \quad \ell_1\text{:}\ \ y := 1 \end{bmatrix}; \quad P_2 :: \begin{bmatrix} m_0\text{:}\ \ \textbf{if } y \neq 1 \textbf{ then} \\ \quad m_1\text{:}\ \ \textbf{await } x = 1 \\ m_2\text{:}\ \ z := 1 \end{bmatrix}$$

Figure 4.5: Program `SIMPLE-SEQ`.

Example: Consider program `SIMPLE-SEQ` in Figure 4.5. Rule `SEQ-SAFE` may be applied to establish

$$M_1; M_2 \ \vDash (\underbrace{y = z = 0}_{\Theta_1^s} \,\wedge\, Env) \rightarrow \square\, \underbrace{z = 1 \ \rightarrow \diamondminus\, x = 1}_{p}$$

where the environment assumption Env ensures that y and z are not modified by the environment:

$$\begin{aligned} Env: &\quad taken(\tau_E) \Rightarrow \mathcal{E} \\ \mathcal{E}: &\quad (y' = y) \wedge (z' = z) \end{aligned}$$

In particular, the premises of rule `SEQ-SAFE` are valid for the following choice of Θ_2, p_1, and p_2:

$$\begin{aligned} \Theta_2: &\quad z = 0 \\ p_1: &\quad y = 1 \Rightarrow \diamondminus\, x = 1 \\ p_2: &\quad z = 1 \Rightarrow (\diamondminus\, x = 1 \vee \boxminus\, y = 1) \end{aligned}$$

Premise QS2 states that $(p_2\, \mathcal{F}\, p_1) \Rightarrow p$ is valid, established previously. ∎

For the case where p is a state formula, take

$$\begin{aligned} p_1: &\quad true \\ p_2: &\quad p \end{aligned}$$

to derive a simpler proof rule:

SEQ − INV

For an assertion Θ_2^s:

SI1.	$M_1 \models (\Theta_1^s \wedge Env) \rightarrow \Box\, p$
SI2.	$M_1 \models (\Theta_1^s \wedge Env) \rightarrow (\Omega_1^p \Rightarrow \Theta_2^s))$
SI3.	$M_2 \models (\Theta_2^s \wedge Env) \rightarrow \Box\, p$
	$M \models (\Theta_1^s \wedge Env) \rightarrow \Box\, p$

The following proof rule may be used to verify safety properties of M, where M is the iteration of M_1 with respect to c.

$$
\boxed{
\begin{array}{l}
\texttt{ITR}-\texttt{SAFE} \\
\text{For a past formula } \varphi\text{:} \\
\quad \begin{array}{ll}
\text{IS1.} & M_f^c \ \vDash \ \Theta \ \to \ \varphi \\
\text{IS2.} & \vDash \ \varphi \ \to \ p \\
\text{IS3.} & \vDash \ \varphi \, \mathcal{F} \, \varphi \Rightarrow \varphi \\
\text{IS4.} & M_f^c \ \vDash (Env \ \wedge \ \varphi) \ \to \ \Box \varphi \\
\text{IS5.} & M_1 \ \vDash (Env \ \wedge \ \varphi) \ \to \ \Box \varphi \\
\hline
 & M \ \vDash (Env \ \wedge \ \Theta) \ \to \ \Box p
\end{array}
\end{array}
}
$$

$$
M :: \left[\begin{array}{l}
\textbf{while } x > 0 \textbf{ do} \\
\quad M_1 :: \left[\begin{array}{ll} \ell_0\text{:} & x := x - 1 \\ \ell_1\text{:} & \end{array}\right]
\end{array}\right]
$$

Figure 4.6: Program DECREASE.

Example: Consider program DECREASE in Figure 4.6. Rule ITR-SAFE may be applied to establish

$$M \ \vDash (\underbrace{x \geq 0}_{\Theta} \ \wedge \ Env) \ \to \ \Box \underbrace{x \geq 0}_{p}$$

where the environment restriction Env ensures that the condition $x > 0$ is not falsified by the environment, i.e.:

$$
\begin{array}{rl}
Env: & taken(\tau_E) \Rightarrow \mathcal{E} \\
\mathcal{E}: & (x > 0) \to (x' > 0)
\end{array}
$$

In particular, the premises of rule ITR-SAFE are valid for:

$$\varphi: \ (x \geq 0) \ \wedge (at\_\ell_1 \to x > 0)$$

■

Proof Rules for Response Properties

SEQ − RESP

For past formulas p_1, p_2, r_1, r_2 and assertions Θ_2^s and ψ:

QR1. $\models (\neg p_2)\, \mathcal{F}\, p_1 \Rightarrow \neg p$

QR2. $\models (r_2\, \mathcal{F}\, r_1) \Rightarrow r$

QR3. $M_1 \models (\Theta_1^s \wedge Env) \rightarrow (\Omega_1^p \Rightarrow p_1 \wedge r_1 \wedge \Theta_2^s)$

QR4. $M_1 \models (\Theta_1^s \wedge Env) \rightarrow (p \Rightarrow \diamondsuit(r \vee (\Omega_1^p \wedge \psi)))$

QR5. $M_2 \models (\Theta_2^s \wedge Env) \rightarrow ((p_2 \vee (\mathit{first} \wedge \psi)) \Rightarrow \diamondsuit\, r_2)$

$$\overline{}$$

$M \models (\Theta_1^s \wedge Env) \rightarrow (p \Rightarrow \diamondsuit\, r)$

Proof:

Let σ be a computation of M satisfying Θ_1^s and Env. Let σ_1 and σ_2 be computations of M_1 and M_2 as described in the proof of Theorem 4.2.1. Assume p holds at position j. If $j \leq |\sigma_1|$, then by SR4 there is a position k, $j \leq k \leq |\sigma_1|$, such that either r holds at k or $k = |\sigma_1|$ and ψ holds at k. In the second case, by QR3, r_1 and Θ_2^s hold at position k. Since Θ_2^s and ψ hold at position k, which is the first state of σ_2, by QR5 there is some k', $k' \geq |\sigma_1|$, such that r_2 holds at position k' of $\sigma_1 \star \sigma_2$ after removing prefix σ_1. By QR2 it follows that k' is an r-position.

Otherwise, $j > |\sigma_1|$. By QR3, p_1, r_1, and Θ_2^s hold at position $|\sigma_1|$. By QR1, p_2 must hold at position j of $\sigma_1 \star \sigma_2$ after removing prefix σ_1, and by QR5, so does $\diamondsuit\, r_2$. Then $\diamondsuit\, r$ holds at j by QR2. ∎

For the case where p and r are state formulas, take

$p_1 :$ *true*

$p_2 :$ p

$r_1 :$ *true*

$r_2 :$ r

to derive a simpler proof rule:

$$
\begin{array}{|l|}
\hline
\texttt{SEQ}-\texttt{SRESP} \\
\text{For assertions } \Theta_2^s \text{ and } \psi\text{:} \\
\quad \text{QSR1. } M_1 \models (\Theta_1^s \wedge Env) \rightarrow (\Omega_1^p \Rightarrow \Theta_2^s) \\
\quad \text{QSR2. } M_1 \models (\Theta_1^s \wedge Env) \rightarrow (p \Rightarrow \Diamond(r \vee (\Omega_1^p \wedge \psi))) \\
\quad \underline{\text{QSR3. } M_2 \models (\Theta_2^s \wedge Env) \rightarrow ((p_2 \vee (\mathit{first} \wedge \psi)) \Rightarrow \Diamond r)} \\
\qquad\qquad M \models (\Theta_1^s \wedge Env) \rightarrow (p \Rightarrow \Diamond r) \\
\hline
\end{array}
$$

The following proof rule may be used to establish response properties when M is the iteration of M_1 with respect to condition c.

$$
\begin{array}{|l|}
\hline
\texttt{ITR}-\texttt{RESP} \\
\text{For a well-founded measure } \delta \text{ over } (A, \prec) \text{ and as-} \\
\text{sertions } \varphi \text{ and } \psi\text{:} \\
\quad \text{IR1. } \ M \ \models (Env \wedge \Theta) \rightarrow \Box\varphi \\
\quad \text{IR2. } \ M_f^c \ \models (Env \wedge \varphi) \rightarrow ((p \vee \psi) \Rightarrow \Diamond(q \vee (\Theta_1^p \wedge \psi))) \\
\quad \text{IR3. } \ M_1 \ \models (Env \wedge \varphi) \rightarrow (p \Rightarrow \Diamond(q \vee (\Omega_1^p \wedge \psi))) \\
\quad \text{IR4. } \ M_1 \ \models (Env \wedge \varphi \wedge \psi \wedge \delta = \alpha) \\
\qquad\qquad\qquad \underline{\rightarrow \Diamond(q \vee (\Omega_1^p \wedge \psi \wedge \delta \prec \alpha))} \\
\qquad\quad M \ \models (Env \wedge \Theta) \rightarrow (p \Rightarrow \Diamond q) \\
\hline
\end{array}
$$

4.3 Floating Composition

4.3.1 Computations of a Transition Module

A sequence of $\mathcal{V}$-states σ is defined to be a *k-computation* of M, where $k \leq |\sigma|$, if it satisfies the following requirements:

- *Initiation* The state s_k is an initial state.
- *Consecution* For every j, where $k \leq j < |\sigma|$:
  - either the state s_{j+1} is a τ-successor of the state s_j for some $\tau \in \mathcal{T}$, i.e., transition τ is *taken* at position j in σ,

  - or the state s_{j+1} is a τ_E-successor of state s_j. In this case, an *environment step* was taken at position j in σ.

- *Justice* If σ is infinite, then for each $\tau \in \mathcal{T}$ it is not the case that τ is continually enabled beyond some point in σ but taken at only finitely many positions in σ.

4.3.2 Floating Temporal Logic

A model for a temporal formula is a pair (σ, j), where σ is a sequence of states and $0 \leq j \leq |\sigma|$. A formula p is *satisfiable* if it holds on some model; it is *valid*, denoted $\models p$, if it holds on all models. A temporal formula p is M-valid, for some transition module M, if $(\sigma, k) \models p$ for every k-computation of M.

Note that, although every valid formula is M-valid, not every deduction rule for general validity is sound for M-validity. In particular, rule `GEN` $p \vdash \square p$ and rule `P-GEN` $p \vdash \boxminus p$ are sound for general validity, but not for M-validity.

4.3.3 Verification

Proof Rules for Safety Properties

`SEQ − SAFE`

For past formula Θ_2^s:

$$\begin{array}{ll} \text{QS1.} & M_1 \models (\Theta_1^s \wedge Env) \rightarrow \square p \\ \text{QS2.} & M_1 \models (\Theta_1^s \wedge Env) \rightarrow (\Omega_1^p \Rightarrow \Theta_2^s) \\ \text{QS3.} & M_2 \models (\Theta_2^s \wedge Env) \rightarrow \square p_2 \\ \hline & M \models (\Theta_1^s \wedge Env) \rightarrow \square p \end{array}$$

Example: Consider program `SIMPLE-SEQ`, presented earlier in Figure 4.5. Rule `SEQ-SAFE` may be applied to establish

$$M_1; M_2 \models (\underbrace{y = z = 0}_{\Theta} \wedge Env) \rightarrow \square \underbrace{z = 1 \rightarrow \diamondminus x = 1}_{p}$$

where the environment assumption Env ensures that y and z are not modified by the environment:

$$\begin{aligned} Env : &\quad taken(\tau_E) \Rightarrow \mathcal{E} \\ \mathcal{E} : &\quad (y' = y) \wedge (z' = z) \end{aligned}$$

In particular, the premises of rule `SEQ-SAFE` are valid for the following choice of Θ_2:

$$\Theta_2 : \; y = 1 \Rightarrow \diamondminus x = 1$$

∎

$$\boxed{\begin{array}{l} \mathtt{ITR-SAFE} \\ \text{For a past formula } \varphi\text{:} \\ \begin{array}{ll} \text{IS1.} & M_f^c \models \Theta \rightarrow \varphi \\ \text{IS2.} & \models \varphi \Rightarrow p \\ \text{IS3.} & M_f^c \models (Env \wedge \varphi) \rightarrow \Box \varphi \\ \text{IS4.} & M_1 \models (Env \wedge \varphi) \rightarrow \Box \varphi \\ \hline & M \models (Env \wedge \Theta) \rightarrow \Box p \end{array} \end{array}}$$

Proof Rules for Response Properties

$$\boxed{\begin{array}{l} \mathtt{SEQ-RESP} \\ \text{For past formula } \Theta_2^s \text{ and assertion } \psi\text{:} \\ \begin{array}{ll} \text{QR1.} & M_1 \models (\Theta_1^s \wedge Env) \rightarrow (\Omega_1^p \Rightarrow \Theta_2^s) \\ \text{QR2.} & M_1 \models (\Theta_1^s \wedge Env) \rightarrow (p \Rightarrow \Diamond(r \vee (\Omega_1^p \wedge \psi))) \\ \text{QR3.} & M_2 \models (\Theta_2^s \wedge Env) \rightarrow ((p \vee (first \wedge \psi)) \Rightarrow \Diamond r) \\ \hline & M \models (\Theta_1^s \wedge Env) \rightarrow (p \Rightarrow \Diamond r) \end{array} \end{array}}$$

`ITR − RESP`

For a well-founded measure δ over $(A, \prec)$ and past formulas φ and ψ:

$$
\begin{array}{lll}
\text{IR1.} & M & \vDash (Env \wedge \Theta) \rightarrow \square\varphi \\
\text{IR2.} & M_f & \vDash (Env \wedge \varphi) \rightarrow ((p \vee \psi) \Rightarrow \diamond(q \vee (\Theta_1^p \wedge \psi))) \\
\text{IR3.} & M_1 & \vDash (Env \wedge \varphi) \rightarrow (p \Rightarrow \diamond(q \vee (\Omega_1^p \wedge \psi))) \\
\text{IR4.} & M_1 & \vDash (Env \wedge \varphi \wedge \psi \wedge \delta = \alpha) \\
 & & \qquad \rightarrow \diamond(q \vee (\Omega_1^p \wedge \psi \wedge \delta \prec \alpha)) \\
\hline
 & M & \vDash (Env \wedge \Theta) \rightarrow (p \Rightarrow \diamond q)
\end{array}
$$

Chapter 5

Real-time Composition

This chapter extends the compositional methods developed earlier to real-time. For simplicity, only parallel composition is considered.

5.1 Real-time Transition Modules

The real-time transition modules defined in this section are based on *timed transition systems*, as presented in [HMP93].

A *real-time transition module* M consists of the following components:

- $V \subseteq \mathcal{V}$: A finite set of variables, partitioned into private variables V^p and shared variables V^s.

- $\mathcal{T}$: A finite set of *transitions*, i.e., assertions over V and V'. A state s' is a τ-*successor* of s if

  $$\langle s, s' \rangle \ \models \ \tau \ \wedge \ \bigwedge_{u \notin V} (u' = u)$$

  For every τ-successor s' of a state s, where $\tau \in \mathcal{T}$, there must be some $u \in V^p$ such that $s'[u] \neq s[u]$. Furthermore, each transition must be *self-disabling*, i.e., each τ must be disabled on every τ-successor of any state s. These conditions are readily satisfied, for instance, by a private control variable that changes, when each transition is taken, to a value which disables the transition.

- Θ: An *initial condition*, i.e., an assertion over V characterizing the initial states of the transition module.
- l: A *lower bound* $l^\tau \in \mathbb{N}$ for each transition τ.
- u: An *upper bound* $u^\tau \in \mathbb{N}$ for each transition τ.

5.1.1 The Environment Transition

The environment transition τ_E is intended to capture every possible behavior of the environment of M. Specifically, a state s' is a τ_E-successor of s if s and s' agree on the private variables of M:

$$\langle s, s' \rangle \models \bigwedge_{u \in V^p} (u' = u)$$

5.1.2 Computations of a Transition Module

A *timed state sequence* ρ may be represented as a pair (σ, T), where $\sigma : s_0, s_1, \ldots$ is an infinite sequence of $\mathcal{V}$-states and $T = T_0, T_1, \ldots$ is a corresponding sequence of time values satisfying the following conditions:

- *Monotonicity* For all $i \geq 0$:

  either $T_{i+1} = T_i$,
  or $T_{i+1} = T_i + 1$ and $s_{i+1} = s_i$.

  The case that time increases is referred to as a *tick step*.
- *Progress* For all $i \geq 0$ there is some $j > i$ such that $T_j > T_i$.

A timed state sequence ρ is defined to be a computation of M if it satisfies the following requirements:

- *Initiation* The state s_0 is an initial state.
- *Consecution* For every j, where $0 \leq j$:
  - either the state s_{j+1} is a τ-successor of the state s_j for some $\tau \in \mathcal{T}$, i.e., transition τ was *taken* at position j in σ,

- or the state s_{j+1} is a τ_E-successor of state s_j, i.e., an *environment step* is taken at position j in σ.

- *Lower bound* No transition $\tau \in \mathcal{T}$ is taken unless it has been continually enabled at least l^τ time units.

- *Upper bound* No transition $\tau \in \mathcal{T}$ may be continually enabled for greater than u^τ time units without being taken.

The set of computations of M is denoted $Comp(M)$.

5.1.3 Parallel Composition

Transition modules M_1 and M_2 may be composed if the private variables of each module are not variables of the other module. In particular:

- $V_1^p \cap V_2 = \emptyset$
- $V_2^p \cap V_1 = \emptyset$
- $\Theta_1 \wedge \Theta_2$ is satisfiable

The parallel composition of M_1 and M_2 yields the transition module M, defined as follows:

- $V = V_1 \cup V_2$

$$V^p = V_1^p \cup V_2^p$$
$$V^s = V_1^s \cup V_2^s$$

- $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$
- $\Theta = \Theta_1 \wedge \Theta_2$
- $l^\tau = l_1^\tau$ if $\tau \in \mathcal{T}_1$, otherwise l_2^τ.
- $u^\tau = u_1^\tau$ if $\tau \in \mathcal{T}_1$, otherwise u_2^τ.

Theorem 5.1.1 *$Comp(M) = Comp(M_1) \cap Comp(M_2)$*

5.2 Metric Temporal Logic

A *metric temporal formula* is constructed out of state formulas, the boolean operators $\neg$ and $\vee$, and the following temporal operators:

$\bigcirc_{\sim c}$ — Next	$\mathcal{U}_{\sim c}$ — Until
$\ominus_{\sim c}$ — Previous	$\mathcal{S}_{\sim c}$ — Since

where $\sim$ is one of $\{<, =, >, \equiv_d\}$, $c \geq 0$, and $d \geq 2$.

Given a timed state sequence $\rho = (\sigma, T)$ and a metric temporal formula p, $(\rho, j) \models p$ denotes that p holds at position j in ρ. For a state formula p,

$$(\rho, j) \models p \Longleftrightarrow s_j \models p$$

That is, p is evaluated locally, using the interpretation given by s_j. The state s_j is a *p-state* if p holds on s_j.

$$
\begin{array}{lcl}
(\rho, j) \models \neg p & \Longleftrightarrow & (\rho, j) \not\models p \\
(\rho, j) \models p \vee q & \Longleftrightarrow & (\rho, j) \models p \text{ or } (\rho, j) \models q \\
(\rho, j) \models \bigcirc_{\sim c}\, p & \Longleftrightarrow & (\rho, j+1) \models p \text{ and } (T_{j+1} - T_j) \sim c \\
(\rho, j) \models p\, \mathcal{U}_{\sim c}\, q & \Longleftrightarrow & \text{for some } k,\ j \leq k, \\
 & & \quad (\rho, k) \models q \text{ and } (T_k - T_j) \sim c \\
 & & \text{and for every } i,\ j \leq i < k,\ (\rho, i) \models p \\
(\rho, j) \models \ominus_{\sim c}\, p & \Longleftrightarrow & j \geq 1 \text{ and } (\rho, j-1) \models p \text{ and } (T_j - T_{j-1}) \sim c \\
(\rho, j) \models p\, \mathcal{S}_{\sim c}\, q & \Longleftrightarrow & \text{for some } k,\ 0 \leq k \leq j, \\
 & & \quad (\rho, k) \models q \text{ and } (T_j - T_k) \sim c \\
 & & \text{and for every } i,\ k < i \leq j,\ (\rho, i) \models p
\end{array}
$$

An unbounded version of $p\,\mathcal{U}\,q$ may be defined as $p\,\mathcal{U}_{=0}\,q \;\vee\; p\,\mathcal{U}_{>0}\,q$, and similarly for an unbounded version of $\mathcal{S}$. Additional temporal operators can be defined as follows:

$$
\begin{array}{ll}
\widetilde{\bigcirc}_{\sim c} p = \neg \bigcirc_{\sim c} \neg p & \text{— Weak next} \\
\Diamond_{\sim c}\, p = \mathit{true}\ \mathcal{U}_{\sim c}\, p & \text{— Eventually} \\
\Box_{\sim c}\, p = \neg \Diamond_{\sim c} \neg p & \text{— Henceforth} \\
p\, \mathcal{W}_{\sim c}\, q = \Box\, p \vee (p\, \mathcal{U}_{\sim c}\, q) & \text{— Waiting-for} \\
\widetilde{\ominus}_{\sim c} p = \neg \ominus_{\sim c} \neg p & \text{— Weak previously} \\
\diamondminus_{\sim c}\, p = \mathit{true}\ \mathcal{S}_{\sim c}\, p & \text{— Sometime in the past} \\
\boxminus_{\sim c}\, p = \neg \diamondminus_{\sim c} \neg p & \text{— Always in the past} \\
p\, \mathcal{B}_{\sim c}\, q = \boxminus\, p \vee (p\, \mathcal{S}_{\sim c}\, q) & \text{— Back-to}
\end{array}
$$

Similarly, unbounded versions of the above operators can be defined from the unbounded $\mathcal{U}$ and $\mathcal{S}$ operators. Note that, since time can change by at most 1 between two adjacent states, it is sufficient to allow $\sim c$ to be $= 0$ or $= 1$ for the immediate operators $\bigcirc_{\sim c}$ and $\ominus_{\sim c}$. Then weak and unbounded versions of $\bigcirc$ and $\ominus$ are defined as follows:

$$
\begin{aligned}
\bigcirc\, p &= \bigcirc_{=0}\, p \vee \bigcirc_{=1}\, p \\
\ominus\, p &= \ominus_{=0}\, p \vee \ominus_{=1}\, p \\
\widetilde{\ominus} p &= \neg \ominus \neg p
\end{aligned}
$$

The formulas *tick* and *ptick* are defined as follows:

$$
\begin{aligned}
\mathit{tick} &: \quad \bigcirc_{=1}\, \mathit{true} \\
\mathit{ptick} &: \quad \ominus_{=1}\, \mathit{true}
\end{aligned}
$$

5.2.1 A Decision Procedure

The following decision procedure follows closely the decision procedure for PTL given in [LPZ85].

The *closure* of a metric temporal formula φ is the smallest set of formulas $Cl(\varphi)$ containing φ and satisfying:

$$
\begin{array}{lcl}
tick, ptick \in Cl(\varphi) & & \\
p \in Cl(\varphi) & \Rightarrow & \neg p \in Cl(\varphi) \\
p \vee q \in Cl(\varphi) & \Rightarrow & p, q \in Cl(\varphi) \\
\bigcirc_{\sim c}\, p \in Cl(\varphi) & \Rightarrow & p, \bigcirc p \in Cl(\varphi) \\
p\,\mathcal{U}_{\sim c}\, q \in Cl(\varphi) & \Rightarrow & p, q, \bigcirc(p\,\mathcal{U}_{\sim c}\, q), \bigcirc(p\,\mathcal{U}_{\sim c-1}\, q) \in Cl(\varphi) \\
\ominus_{\sim c}\, p \in Cl(\varphi) & \Rightarrow & p, \ominus p \in Cl(\varphi) \\
p\,\mathcal{S}_{\sim c}\, q \in Cl(\varphi) & \Rightarrow & p, q, \ominus(p\,\mathcal{S}_{\sim c}\, q), \ominus(p\,\mathcal{S}_{\sim c-1}\, q) \in Cl(\varphi)
\end{array}
$$

where:

$$
\begin{array}{ll}
p\,\mathcal{U}_{<-1}\, q = \mathit{false} & p\,\mathcal{S}_{<-1}\, q = \mathit{false} \\
p\,\mathcal{U}_{=-1}\, q = \mathit{false} & p\,\mathcal{S}_{=-1}\, q = \mathit{false} \\
p\,\mathcal{U}_{>-1}\, q = p\,\mathcal{U}\, q & p\,\mathcal{S}_{>-1}\, q = p\,\mathcal{S}\, q \\
p\,\mathcal{U}_{\equiv_d -1}\, q = p\,\mathcal{U}_{\equiv_d d-1}\, q & p\,\mathcal{S}_{\equiv_d -1}\, q = p\,\mathcal{S}_{\equiv_d d-1}\, q
\end{array}
$$

An *atom* is a set of formulas $A \subset Cl(\varphi)$ such that:

$$
\begin{array}{lcl}
true \in A & & \\
p \in A & \Longleftrightarrow & \neg p \notin A \\
p \vee q \in A & \Longleftrightarrow & p \in A \text{ or } q \in A \\
\bigcirc_{=0}\, p \in A & \Longleftrightarrow & \neg tick, \bigcirc p \in A \\
\bigcirc_{=1}\, p \in A & \Longleftrightarrow & tick, \bigcirc p \in A \\
p\,\mathcal{U}_{<0}\, q \notin A & & \\
p\,\mathcal{U}_{<c+1}\, q \in A & \Longleftrightarrow & q \in A \text{ or } p, \neg tick, \bigcirc(p\,\mathcal{U}_{<c+1}\, q) \in A \\
 & & \qquad \text{or } p, tick, \bigcirc(p\,\mathcal{U}_{<c}\, q) \in A \\
p\,\mathcal{U}_{=0}\, q \in A & \Longleftrightarrow & q \in A \text{ or } p, \neg tick, \bigcirc(p\,\mathcal{U}_{=0}\, q) \in A \\
p\,\mathcal{U}_{=c+1}\, q \in A & \Longleftrightarrow & p, \neg tick, \bigcirc(p\,\mathcal{U}_{=c+1}\, q) \in A \text{ or } p, tick, \bigcirc(p\,\mathcal{U}_{=c}\, q) \in A \\
p\,\mathcal{U}_{>0}\, q \in A & \Longleftrightarrow & p, \neg tick, \bigcirc(p\,\mathcal{U}_{>0}\, q) \in A \text{ or } p, tick, \bigcirc(p\,\mathcal{U}\, q) \in A \\
p\,\mathcal{U}_{>c+1}\, q \in A & \Longleftrightarrow & p, \neg tick, \bigcirc(p\,\mathcal{U}_{>c+1}\, q) \in A \text{ or } p, tick, \bigcirc(p\,\mathcal{U}_{>c}\, q) \in A \\
p\,\mathcal{U}_{\equiv_d 0}\, q \in A & \Longleftrightarrow & q \in A \text{ or } p, \neg tick, \bigcirc(p\,\mathcal{U}_{\equiv_d 0}\, q) \in A \\
 & & \qquad \text{or } p, tick, \bigcirc(p\,\mathcal{U}_{\equiv_d d-1}\, q) \in A \\
p\,\mathcal{U}_{\equiv_d c+1}\, q \in A & \Longleftrightarrow & p, \neg tick, \bigcirc(p\,\mathcal{U}_{\equiv_d c+1}\, q) \in A \text{ or } p, tick, \bigcirc(p\,\mathcal{U}_{\equiv_d c}\, q) \in A \\
\ominus_{=0}\, p \in A & \Longleftrightarrow & \neg ptick, \ominus p \in A \\
\ominus_{=1}\, p \in A & \Longleftrightarrow & ptick, \ominus p \in A
\end{array}
$$

$$
\begin{array}{lll}
p\,\mathcal{S}_{<0}\,q \notin A & & \\
p\,\mathcal{S}_{<c+1}\,q \in A & \iff & q \in A \text{ or } p, \neg ptick, \ominus(p\,\mathcal{S}_{<c+1}\,q) \in A \\
 & & \quad \text{or } p, ptick, \ominus(p\,\mathcal{S}_{<c}\,q) \in A \\
p\,\mathcal{S}_{=0}\,q \in A & \iff & q \in A \text{ or } p, \neg ptick, \ominus(p\,\mathcal{S}_{=0}\,q) \in A \\
p\,\mathcal{S}_{=c+1}\,q \in A & \iff & p, \neg ptick, \ominus(p\,\mathcal{S}_{=c+1}\,q) \in A \text{ or } p, ptick, \ominus(p\,\mathcal{S}_{=c}\,q) \in A \\
p\,\mathcal{S}_{>0}\,q \in A & \iff & p, \neg ptick, \ominus(p\,\mathcal{S}_{>0}\,q) \in A \text{ or } p, ptick, \ominus(p\,\mathcal{S}\,q) \in A \\
p\,\mathcal{S}_{>c+1}\,q \in A & \iff & p, \neg ptick, \ominus(p\,\mathcal{S}_{>c+1}\,q) \in A \text{ or } p, ptick, \ominus(p\,\mathcal{S}_{>c}\,q) \in A \\
p\,\mathcal{S}_{\equiv_d 0}\,q \in A & \iff & q \in A \text{ or } p, \neg ptick, \ominus(p\,\mathcal{S}_{\equiv_d 0}\,q) \in A \\
 & & \quad \text{or } p, ptick, \ominus(p\,\mathcal{S}_{\equiv_d d-1}\,q) \in A \\
p\,\mathcal{S}_{\equiv_d c+1}\,q \in A & \iff & p, \neg ptick, \ominus(p\,\mathcal{S}_{\equiv_d c+1}\,q) \in A \text{ or } p, ptick, \ominus(p\,\mathcal{S}_{\equiv_d c}\,q) \in A
\end{array}
$$

The set of atoms for φ is denoted $Atoms(\varphi)$.

A formula is said to be *elementary* if it is a proposition, has $\bigcirc$ or $\ominus$ as its main connective, or is either *tick* or *ptick*. Let $ECl(\varphi) \subseteq Cl(\varphi)$ denote the *elementary closure* of φ, consisting of the elementary formulas in $Cl(\varphi)$.

Note that there is a one-to-one correspondence between the set of atoms and the set $\{E \subseteq ECl(\varphi)\}$. In particular, for every set of elementary formulas $E \subseteq ECl(\varphi)$, there is an atom containing every elementary formula in E and the negation of every elementary formula not in E.

The initial tableau $\mathcal{T}_0 = (W_0, R_0)$ is a graph whose nodes are exactly the atoms of φ. There is an edge $(A_1, A_2) \in R_0$ if the following conditions hold:

- $tick \in A_1 \iff ptick \in A_2$
- $\bigcirc p \in A_1 \iff p \in A_2$
- $p \in A_1 \iff \ominus p \in A_2$

A path $A_0, A_1, A_2, \ldots$ in $\mathcal{T}_i$ *fulfills* φ if the following conditions hold:

- A_0 is *initial*, i.e., does not contain any formula of the form $\ominus p$,
- $\varphi \in A_0$, and
- for every $p\,\mathcal{U}\,q \in A_j$, there is some $k \geq j$ such that $q \in A_k$.

Proposition 5.2.1 *The formula φ is satisfiable if and only if there is a path in $\mathcal{T}_0$ that fulfills φ.*

A sequence of progressively smaller tableaus $\mathcal{T}_1, \mathcal{T}_2, \ldots$ may be obtained from $\mathcal{T}_0$ by repeatedly removing atoms which cannot participate in paths fulfilling φ. In particular, $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by identifying and removing a maximal strongly connected subgraph $\mathcal{C}$ in $\mathcal{T}_i$ that satisfies one of the following conditions:

- $\mathcal{C}$ contains no incoming edges and no initial atoms, or
- $\mathcal{C}$ has no outgoing edges and is not self-fulfilling, or
- $\mathcal{C}$ is a single atom with no successors

where a strongly connected graph $\mathcal{C}$ is said to be *self-fulfilling* if, for every $p \, \mathcal{U} \, q \in A_1 \in \mathcal{C}$, there is some $A_2 \in \mathcal{C}$ such that $q \in A_2$.

Proposition 5.2.2 *There is a path in $\mathcal{T}_{i+1}$ that fulfills φ if and only if there is a path in $\mathcal{T}_i$ that fulfills φ.*

When $\mathcal{T}_k$ is empty or cannot be reduced any further, the following proposition holds.

Proposition 5.2.3 *The formula φ is satisfiable if and only if there is an initial atom in $\mathcal{T}_k$ containing φ.*

5.2.2 A Deductive System

The deductive system presented below is very similar to the deductive system for PTL presented in [MP92]. In fact, only axioms FX6 and PX6 have been changed, and axioms FX9, PX9, and FX10 have been added.

Future Axioms

FX0. $\Box\, p \;\; \rightarrow \;\; p$

FX1. $\neg \bigcirc p \;\; \Leftrightarrow \;\; \bigcirc \neg p$

FX2. $\bigcirc(p \; \rightarrow \; q) \;\; \Leftrightarrow (\bigcirc p \rightarrow \bigcirc q)$

FX3. $\Box(p \; \rightarrow \; q) \;\; \Rightarrow (\Box\, p \rightarrow \Box\, q)$

FX4. $\Box\, p \;\; \rightarrow \;\; \Box \bigcirc p$

FX5. $(p \Rightarrow \bigcirc p) \;\; \rightarrow \;\; (p \Rightarrow \Box\, p)$

FX6. $\Box\, \neg(p\, \mathcal{U}_{<0}\, q)$

$$\begin{aligned}
p\,\mathcal{U}_{<c+1}\, q \quad & \Leftrightarrow \;\; q \vee (p \wedge (\bigcirc_{=0} (p\,\mathcal{U}_{<c+1}\, q) \\
& \qquad\qquad \vee \bigcirc_{=1} (p\,\mathcal{U}_{<c}\, q))) \\
p\,\mathcal{U}_{=0}\, q \quad & \Leftrightarrow q \vee (p \wedge \bigcirc_{=0} (p\,\mathcal{U}_{=0}\, q)) \\
p\,\mathcal{U}_{=c+1}\, q \quad & \Leftrightarrow \;\; p \wedge (\bigcirc_{=0} (p\,\mathcal{U}_{=c+1}\, q) \\
& \qquad \vee \bigcirc_{=1} (p\,\mathcal{U}_{=c}\, q)) \\
p\,\mathcal{U}_{>0}\, q \quad & \Leftrightarrow \;\; p \wedge (\bigcirc_{=0} (p\,\mathcal{U}_{>0}\, q) \\
& \qquad \vee \bigcirc_{=1} (p\,\mathcal{U}\, q)) \\
p\,\mathcal{U}_{>c+1}\, q \quad & \Leftrightarrow \;\; p \wedge (\bigcirc_{=0} (p\,\mathcal{U}_{>c+1}\, q) \\
& \qquad \vee \bigcirc_{=1} (p\,\mathcal{U}_{>c}\, q)) \\
p\,\mathcal{U}_{\equiv_d 0}\, q \quad & \Leftrightarrow \;\; q \vee (p \wedge (\bigcirc_{=0} (p\,\mathcal{U}_{\equiv_d 0}\, q) \\
& \qquad\qquad \vee \bigcirc_{=1} (p\,\mathcal{U}_{\equiv_d d-1}\, q))) \\
p\,\mathcal{U}_{\equiv_d c+1}\, q \quad & \Leftrightarrow \;\; p \wedge (\bigcirc_{=0} (p\,\mathcal{U}_{\equiv_d c+1}\, q) \\
& \qquad \vee \bigcirc_{=1} (p\,\mathcal{U}_{\equiv_d c}\, q))
\end{aligned}$$

FX7. $\Box\, p \;\; \Rightarrow \;\; p\,\mathcal{W}\, q$

FX8. $p \;\; \Rightarrow \;\; \bigcirc \ominus p$

FX9. $\bigcirc_{=0}\, p \;\; \Rightarrow \;\; \neg\bigcirc_{=1}\, q$

FX10. $tick \;\; \Leftrightarrow \;\; \bigcirc\, ptick$

Past Axioms

PX1. $\ominus\, p \;\; \Rightarrow \;\; \widetilde{\ominus} p$

PX2. $\widetilde{\ominus}(p \; \rightarrow \; q) \;\; \Leftrightarrow (\widetilde{\ominus} p \rightarrow \widetilde{\ominus} q)$

PX3. $\boxminus(p \; \rightarrow \; q) \;\; \Rightarrow (\boxminus\, p \rightarrow \boxminus\, q)$

PX4. $\Box p \;\rightarrow\; \Box \widetilde{\ominus} p$

PX5. $(p \Rightarrow \widetilde{\ominus} p) \;\rightarrow\; (p \Rightarrow \boxminus p)$

PX6. $\Box \neg (p \,\mathcal{S}_{<0}\, q)$

$$
\begin{array}{lll}
p \,\mathcal{S}_{<c+1}\, q & \Leftrightarrow & q \vee (p \wedge (\ominus_{=0} (p \,\mathcal{S}_{<c+1}\, q) \\
 & & \qquad \vee \ominus_{=1} (p \,\mathcal{S}_{<c}\, q))) \\
p \,\mathcal{S}_{=0}\, q & \Leftrightarrow & q \vee (p \wedge \ominus_{=0} (p \,\mathcal{S}_{=0}\, q)) \\
p \,\mathcal{S}_{=c+1}\, q & \Leftrightarrow & p \wedge (\ominus_{=0} (p \,\mathcal{S}_{=c+1}\, q) \\
 & & \quad \vee \ominus_{=1} (p \,\mathcal{S}_{=c}\, q)) \\
p \,\mathcal{S}_{>0}\, q & \Leftrightarrow & p \wedge (\ominus_{=0} (p \,\mathcal{S}_{>0}\, q) \\
 & & \quad \vee \ominus_{=1} (p \,\mathcal{S}\, q)) \\
p \,\mathcal{S}_{>c+1}\, q & \Leftrightarrow & p \wedge (\ominus_{=0} (p \,\mathcal{S}_{>c+1}\, q) \\
 & & \quad \vee \ominus_{=1} (p \,\mathcal{S}_{>c}\, q)) \\
p \,\mathcal{S}_{\equiv_d 0}\, q & \Leftrightarrow & q \vee (p \wedge (\ominus_{=0} (p \,\mathcal{S}_{\equiv_d 0}\, q) \\
 & & \qquad \vee \ominus_{=1} (p \,\mathcal{S}_{\equiv_d d-1}\, q))) \\
p \,\mathcal{S}_{\equiv_d c+1}\, q & \Leftrightarrow & p \wedge (\ominus_{=0} (p \,\mathcal{S}_{\equiv_d c+1}\, q) \\
 & & \quad \vee \ominus_{=1} (p \,\mathcal{S}_{\equiv_d c}\, q))
\end{array}
$$

PX7. $\widetilde{\ominus} \mathit{false}$

PX8. $p \Rightarrow \widetilde{\ominus} \bigcirc p$

PX9. $\ominus_{=0}\, p \;\Rightarrow\; \neg \ominus_{=1}\, q$

TGEN. $\Box p$ for a valid state formula p

The deductive system uses the proof rules *modus ponens*

$$\text{MP}: \quad p \rightarrow q,\, p \;\vdash\; q$$

and *instantiation*

$$\text{INST}: \quad p \;\vdash\; p[\alpha]$$

where α is a replacement for the sentence symbols in p.

Completeness

For a set of formulas $C \subset Cl(\varphi)$, let $\widehat{C}$ denote the conjunction of formulas in C.

Assume that φ is valid. Construct the tableau $\mathcal{T}_0 = (W_0, R_0)$ for $\neg\varphi$ as described above.

Lemma 5.2.4 $\vdash \Box \bigvee_{A \in W_0} \widehat{A}$

Proof:

Establish $\vdash \widehat{B} \Rightarrow \widehat{A}$ by induction, for any atom A. Consider $\psi \in A - B$. Assume $\vdash \widehat{B} \Rightarrow \widehat{C}$, where C includes p or $\neg p$ for every $p \in Cl(\psi)$. Let X be the top-level operator of ψ, ignoring any outermost negation. Since $\varphi \notin B$, X cannot be $\bigcirc$ or $\circleddash$. For the case where X is disjunction, $\vdash \widehat{C} \Rightarrow \psi$ by EMP. Otherwise, let $\Box\chi$ by the appropriate axiom from FX1, FX6, PX1, or PX6 for X. Then $\vdash \chi \Rightarrow (\widehat{C} \rightarrow \psi)$ by TGI, and $\vdash \widehat{C} \Rightarrow \psi$ by EMP. It follows that $\vdash \widehat{B} \Rightarrow \widehat{A}$. Finally, observe $\vdash \Box \bigvee_{A \in W_0} \widehat{B}$ by TGI. ∎

Lemma 5.2.5 *For each* $A_2 \in W_0$*:*

$$\vdash \widehat{A_2} \Rightarrow \left[\left(\widetilde{\circleddash} \bigvee_{(A_1, A_2) \in R_0} \widehat{A_1} \right) \wedge \left(\bigcirc \bigvee_{(A_2, A_3) \in R_0} \widehat{A_3} \right) \right]$$

Proof:

First, observe $\vdash \widehat{A_2} \Rightarrow \left[\left(\widetilde{\circleddash} \bigvee_{A_1 \in W_0} \widehat{A_1} \right) \wedge \left(\bigcirc \bigvee_{A_3 \in W_0} \widehat{A_3} \right) \right]$ by FX4, PX4, and Lemma 5.2.4. Then, by FX2, FX4, PX2, and PX4, it suffices to show $\vdash \widehat{A_2} \Rightarrow \widetilde{\circleddash} \neg \widehat{A_1}$ and $\vdash \widehat{A_2} \Rightarrow \bigcirc \neg \widehat{A_3}$ for every $(A_1, A_2), (A_2, A_3) \notin R_0$, which follow from FX8 and PX8. ∎

Lemma 5.2.6 *For each* $A_2 \in W_0$*:*

$$\vdash \widehat{A_2} \Rightarrow \left[\left(\boxminus \bigvee_{(A_1, A_2) \in R_0} \widehat{A_1} \right) \wedge \left(\Box \bigvee_{(A_2, A_3) \in R_0} \widehat{A_3} \right) \right]$$

Proof:

From FX5, PX5, and Lemma 5.2.5. ∎

The following lemmas hold by induction on the sequence of tableaus $\mathcal{T}_0, \mathcal{T}_1, \ldots$.

Lemma 5.2.7 $\vdash \Box \neg \widehat{A}$ *for each* $A \notin \mathcal{T}_i$

Lemma 5.2.8 *For each* $A_2 \in W_i$*:*

$$\vdash \widehat{A_2} \Rightarrow \left[\left(\widetilde{\circleddash} \bigvee_{(A_1,A_2)\in R_i} \widehat{A_1} \right) \wedge \left(\bigcirc \bigvee_{(A_2,A_3)\in R_i} \widehat{A_3} \right) \right]$$

Lemma 5.2.9 *For each* $A_2 \in W_i$*:*

$$\vdash \widehat{A_2} \Rightarrow \left[\left(\boxminus \bigvee_{(A_1,A_2)\in R_i} \widehat{A_1} \right) \wedge \left(\Box \bigvee_{(A_2,A_3)\in R_i} \widehat{A_3} \right) \right]$$

Let $\mathcal{T}_k$ be the final tableau. Observe that $\vdash \Box \bigvee_{A\in W_k} \widehat{A}$ follows from Lemmas 5.2.4 and 5.2.7, so by rule PAR, $\vdash \bigvee_{A\in W_k} \widehat{A}$. Furthermore, $\vdash \neg\widehat{A}$ for any noninitial atom A, so $\vdash \bigvee_{A\in W_k^0} \widehat{A}$ for the set of initial atoms $W_k^0 \subseteq W_k$. Since φ is valid, $\neg\varphi$ is not satisfiable, and there cannot be any initial atoms in $\mathcal{T}_k$ containing $\neg\varphi$. Therefore φ is contained in every initial atom, yielding $\vdash (\bigvee_{A\in W_k^0} \widehat{A}) \rightarrow \varphi$, which leads to $\vdash \varphi$.

5.3 Verification

5.3.1 The Environment Restriction

The environment transition of a real-time transition module allows arbitrary interference by the environment. As before, an *environment restriction Env* is a formula of the form:

$$\mathit{Env}: \quad \mathit{taken}(\tau_E) \Rightarrow \underbrace{\bigwedge_{i=1}^{m} (\varphi_i \rightarrow \psi_i')}_{\mathcal{E}}$$

where φ_i and ψ_i may now be past MTL formulas, rather than simply assertions.

5.3.2 Global Verification

In order to avoid the use of $\mathit{tick}: \bigcirc_{=1} \mathit{true}$, which is not a past formula, the following axiom is introduced for the new propositional symbol *Tick*:

$\texttt{TICK}: \quad \mathit{Tick} \Leftrightarrow \mathit{tick}$

The bounded monotonicity condition for computations of a real-time transition module corresponds to the following axiom:

$$\texttt{BMON}: \quad Tick \Rightarrow \bigwedge_{u \in \mathcal{V}} u' = u$$

The upper bound condition corresponds to the following axiom:

$$\texttt{UPPER}: \quad \Box \diamondsuit_{\leq u^{\tau}} (\neg En(\tau) \lor \mathit{first}) \text{for each } \tau \in \mathcal{T}$$

The *primed version* of a past formula p is defined, as before, to be a past formula p' such that p holds in the next state of a computation iff p' holds in the current state. For instance:

$$\begin{aligned}
(\circleddash_{=0}\, p)' &= p \land \neg Tick \\
(\circleddash_{=1}\, p)' &= p \land Tick \\
(p\, \mathcal{S}_{\leq c+1}\, q)' &= q' \lor (p' \land Tick \land p\, \mathcal{S}_{\leq c}\, q) \\
&\quad \lor (p' \land \neg Tick \land p\, \mathcal{S}_{\leq c+1}\, q)
\end{aligned}$$

The verification condition of a transition τ with respect to precondition p and postcondition q, where p and q are past formulas, is defined as follows:

$$\{p\}\tau\{q\}: \quad (p \land \tau \land \bigwedge_{u \notin V} (u' = u) \land \neg Tick \land \boxminus_{<L} En(\tau)) \Rightarrow q'$$

For a set of transitions $\mathcal{T}$, $\{p\}\mathcal{T}\{q\}$ is taken to be the conjunction of verification conditions $\{p\}\tau\{q\}$ for each $\tau \in \mathcal{T}$.

Given the environment restriction $Env : \ taken(\tau_E) \Rightarrow \mathcal{E}$, the environment verification condition is given by:

$$\{p\}\mathcal{E}\{q\}: \quad (p \land \mathcal{E} \land \bigwedge_{u \in V^p} (u' = u) \land \neg Tick) \Rightarrow q'$$

Finally, the clock verification condition is given by:

$$\{p\}clock\{q\}: \quad (p \land Tick) \Rightarrow q'$$

Then the following proof rule is sound and complete for establishing safety properties of a real-time transition module M.

R − SAFE
For a past MTL formula φ:
RS1. $\Theta \rightarrow \varphi$
RS2. $\varphi \Rightarrow p$
RS3. $\{\varphi\}\mathcal{T}\{\varphi\}$
RS4. $\{\varphi\}\mathcal{E}\{\varphi\}$
RS5. $\{\varphi\}\mathit{clock}\{\varphi\}$
$M \models \mathit{Env} \rightarrow \Box\, p$

$$M_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{while } x \neq 1 \textbf{ do} \\ \quad \left[\begin{array}{l} \ell_3\text{: } \textbf{await } x = 0 \\ \ell_4\text{: } x := 1 \\ \ell_5\text{: } \textbf{skip} \\ \ell_6\text{: } \textbf{if } x = 1 \textbf{ then} \\ \quad \ell_7\text{: } \textbf{critical} \end{array}\right] \\ \ell_8\text{: } x := 0 \end{array}\right] \end{array}\right] \quad || \quad M_2 :: \left[\begin{array}{l} m_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1\text{: } \textbf{noncritical} \\ m_2\text{: } \textbf{while } x \neq 2 \textbf{ do} \\ \quad \left[\begin{array}{l} m_3\text{: } \textbf{await } x = 0 \\ m_4\text{: } x := 2 \\ m_5\text{: } \textbf{skip} \\ m_6\text{: } \textbf{if } x = 2 \textbf{ then} \\ \quad m_7\text{: } \textbf{critical} \end{array}\right] \\ m_8\text{: } x := 0 \end{array}\right] \end{array}\right]$$

Figure 5.1: Fischer's mutual exclusion algorithm.

Example: Consider Fischer's mutual exclusion algorithm, presented in Figure 5.1. Assume uniform time bounds L and U for each statement, except the noncritical regions ℓ_1 and m_1, which both have lower and upper bounds $3 \cdot L$ and ∞. Further assume that $2 \cdot L > U$. Ignoring process M_2 for a moment, it is possible to establish

$$M_1 \models \mathit{Env} \rightarrow \Box\, \underbrace{\mathit{at\_}\ell_{7,8} \rightarrow x = 1}_{p}$$

where the environment assumption Env is given as:

$$Env: \quad taken(\tau_E) \;\Rightarrow\; \mathcal{E}_1^1 \,\wedge\, \mathcal{E}_2^2$$

$$\mathcal{E}_1^1: \quad x' = x \,\vee\, x' = 0 \,\vee\, x' = 2$$

stating that the environment of M_1 either does not change x, or changes x to 0 or 2.

$$\mathcal{E}_1^2: \quad \boxminus_{\leq U}\, x = 1 \;\rightarrow\; x' = 1$$

stating that, if $x = 1$ has held through at least the last U time units, then the environment does not change x from 1.

The strengthening assertion φ in rule `R-SAFE` is taken to be:

$$\begin{aligned}\varphi: \quad & (at\_\ell_5 \,\wedge\, x = 1) \rightarrow (x = 1)\, \mathcal{B}\; at\_\ell_4 \\ & \wedge\, (at\_\ell_6 \,\wedge\, x = 1) \rightarrow (x = 1)\, \mathcal{B}\; \boxminus_{<L}\, (at\_\ell_5 \,\wedge\, x = 1) \\ & \wedge\; at\_\ell_{7,8} \rightarrow \boxminus_{<2L}\, x = 1\end{aligned}$$

It is not difficult to see that premises RS1, RS2 and RS5 are valid for this choice of φ. Premise RS4, stating that each environment step preserves φ, is valid by the assumption $2 \cdot L > U$. In particular, $\boxminus_{<2\cdot L}\, x = 1 \;\Rightarrow\; \boxminus_{\leq U}\, x = 1$ is valid, so the clause $\mathcal{E}_1^2$ in the environment restriction ensures that the environment of M_1 does not change x from 1 when M_1 is in its critical section.

For premise RS3, the difficult cases to consider are τ_{ℓ_4}, τ_{ℓ_5} and τ_{ℓ_6}. Transition τ_{ℓ_4} implies:

$$at\_\ell_4 \,\wedge\, (at\_\ell_5)' \,\wedge\, x' = 1$$

It follows that

$$(x = 1\; \mathcal{B}\; at\_\ell_4)': \quad (at\_\ell_4)' \,\vee\, (x' = 1 \,\wedge\, x = 1\; \mathcal{B}\; at\_\ell_4)$$

holds, so the verification condition for τ_{ℓ_4} is valid. Now consider the verification condition for τ_{ℓ_5}. If $x \neq 1$, then φ' clearly holds. Otherwise, it follows from $x = 1$, φ, and τ_{ℓ_5} that $x' = 1$ and $x = 1\,\mathcal{B}\; at\_\ell_4$ hold, and it suffices to establish:

$$\begin{array}{l} (x = 1\,\mathcal{B}\,\boxminus_{<L}\,(at\_\ell_5 \;\wedge\; x = 1))' : \\ \qquad \cdots \vee (x' = 1 \;\wedge\; \boxminus_{<L}\,(at\_\ell_5 \;\wedge\; x = 1)) \end{array}$$

Then observe that

$$(\boxminus_{<L}\;\underbrace{at\_\ell_5}_{En(\tau_{\ell_5})}\;\wedge (x = 1)\,\mathcal{B}\; at\_\ell_4) \;\Rightarrow\; \boxminus_{<L}\,(at\_\ell_5 \;\wedge\; x = 1)$$

is easily established to be modularly valid for M_1, thereby proving the verification condition for τ_{ℓ_5}. The verification condition for τ_{ℓ_6} is likewise established by observing that the lower bound on τ_{ℓ_6}, combined with $x = 1\,\mathcal{B}\,\boxminus_{<L}\,(at\_\ell_5 \;\wedge\; x = 1)$, entails $\boxminus_{<2L}\; x = 1$. ▪

Rule `R-SAFE` is complete for proving safety properties. In particular, this includes bounded response formulas of the form $p \Rightarrow \Diamond_{\leq U}\, q$ and minimal separation formulas of the form $p \Rightarrow q\,\mathcal{W}_{>L}\, r$, because, according to the following proposition, these formulas may be rewritten as $\Box\,\varphi$ for a past formula φ.

Proposition 5.3.1

(a) $p \Rightarrow \Diamond_{\leq U}\, q \quad \sim \quad \Box\,\neg((\neg q)\,\widehat{\mathcal{S}}_{>U}\,(p \;\wedge\; \neg q))$

(b) $p \Rightarrow q\,\mathcal{W}_{>L}\, r \quad \sim \quad \Box((\neg q) \rightarrow (\neg p)\,\mathcal{B}\,(r \;\wedge\; \boxminus_{\leq L}\,\neg p))$

Proof:

(a) Assume $p \Rightarrow \Diamond_{\leq U}\, q$, but suppose $(\neg q)\,\widehat{\mathcal{S}}_{>U}\,(p \;\wedge\; \neg q)$ holds for some j. There is some position $i < j$ such that p holds at i, $\neg q$ holds for each k, where $i \leq k < j$, and $T_j - T_i > U$. Clearly $p \rightarrow \Diamond_{\leq U}\, q$ does not hold at i, which is a contradiction.

Conversely, suppose p holds at i but there is no q-position j such that $T_j - T_i \leq U$. Let k be the first position such that $T_k - T_i > U$. Observe that $(\neg q)\,\hat{\mathcal{S}}_{>U}\,(p \,\wedge\, \neg q)$ holds at k, so $\Box\,\neg((\neg q)\,\hat{\mathcal{S}}_{>U}\,(p \,\wedge\, \neg q))$ must be false.

Note that, when q is a state formula, i.e., not instantiatable by rule `INST`, the strict since operator $\hat{\mathcal{S}}$ may be replaced by $\mathcal{S}$.

(b) Suppose there is some $(\neg p)$-position j such that $(\neg p)\,\mathcal{B}\,(r \,\wedge\, \boxminus_{\leq L}\,\neg p)$ is false. Choose the maximal p-position i such that $i \leq j$ and for all $k \in [i,j]$, $r \,\wedge\, \boxminus_{\leq L}\,\neg p$ is false. If $T_j - T_i \leq L$, then $p \Rightarrow q\,\mathcal{W}_{>L}\,r$ is clearly false at i. Otherwise, observe that $\boxminus_{\leq L}\,\neg p$ holds for each $k \in [i,j]$ such that $T_k - T_i > L$, so r must be false for each such k. In other words, there is no r-position $k \in [i,j]$ such that $T_k - T_i > L$, so $p \rightarrow q\,\mathcal{W} > Lr$ is false at i.

Conversely, suppose there is some p-position i such that $q\,\mathcal{W}_{>L}\,r$ is false. If there is some $(\neg q)$-position j such that $T_j - T_i \leq L$, then clearly $(\neg p)\,\mathcal{B}\,(r \,\wedge\, \boxminus_{\leq L}\,\neg p)$ must also be false at j. Otherwise, let j be the minimal $(\neg q)$-position such that $T_j - T_i > L$, and observe that there is no r-position k such that $T_k - T_i > L$ and $k \in [i,j]$. It follows that $T_k - T_i \leq L$ for any r-position $k \in [i,j]$, and consequently $\boxminus_{\leq L}\,\neg p$ must be false for any r-position $k \in [i,j]$. ∎

Although rule `R-SAFE` is complete for safety properties, the following rules are frequently more natural to apply.

`MINSEP`
For a past MTL formula φ and transition τ:
MS1. $p \;\Rightarrow\; \neg En(\tau)$
MS2. $p \;\Rightarrow\; \varphi$
MS3. $\varphi \;\Rightarrow\; q$
MS4. $\{\varphi\}\mathcal{T} - \tau\{\varphi\}$
MS5. $\varphi \,\wedge\, En(\tau) \;\Rightarrow\; r$
$p \;\Rightarrow\; q\,\mathcal{W}_{\geq l_\tau}\,r$

Example: Returning to the previous example, observe that rule `MINSEP` may be used to develop a more gradual proof of the property:

$$M_1 \models Env \rightarrow (at\_\ell_{7,8} \Rightarrow x = 1)$$

First observe that

$$(at\_\ell_7 \land x = 1) \Rightarrow x = 1 \,\mathcal{B}\, at\_\ell_4 \tag{5.1}$$

is valid for the underlying untimed transition module corresponding to M_1; since real-time transition modules are a conservative extension of untimed transition modules [HMP93], it follows that (5.1) is valid for M_1 as well. Therefore, it suffices to establish:

$$M_1 \models at\_\ell_{7,8} \Rightarrow \boxminus_{<2L} \neg at\_\ell_4$$

or, alternatively,

$$M_1 \models at\_\ell_4 \Rightarrow \square_{<2L} \neg at\_\ell_{7,8}$$

which follows from

$$M_1 \models at\_\ell_4 \Rightarrow at\_\ell_{4,5}\, \mathcal{W}_{\geq L}\, at\_\ell_5$$
$$M_1 \models at\_\ell_5 \Rightarrow at\_\ell_{5,6}\, \mathcal{W}_{\geq L}\, at\_\ell_6$$

The first of these is easily established using rule `MINSEP` by choosing $\varphi : at\_\ell_{4,5}$ and $\tau : \tau_{\ell_5}$; the second is established by choosing $\varphi : at\_\ell_{5,6}$ and $\tau : \tau_{\ell_6}$. ▟

`BRESP`

For past formulas $\varphi = \bigvee_{i=0}^{U} \varphi_i$:

BR1. $p \Rightarrow q \lor \varphi$

BR2. $\{\varphi_i\}\mathcal{T}\{q \lor \bigvee_{j\leq i} \varphi_j\}$

BR3. $\{\varphi_i\}\mathcal{E}\{q \lor \bigvee_{j\leq i} \varphi_j\}$

BR4. $\{\varphi_i\}clock\{q \lor \bigvee_{j<i} \varphi_j\}$

BR5. $\varphi_0 \land Tick \Rightarrow q$

$p \Rightarrow \Diamond_{\leq U}\, q$

5.3.3 Compositional Verification

The same proof rule used for compositional verification in the untimed case is used for real-time verification. Recall, however, that the environment restrictions Env, Env_1, $\ldots$, Env_N may now include past MTL formulas. In the following, M is obtained as the parallel composition of transition modules $M_1, \ldots, M_N$.

PAR

For formulas φ_1, $\ldots$, φ_N and environment restrictions Env_1, $\ldots$, Env_N:

P1.	$M_i \models Env_i \rightarrow \varphi_i$
P2.	$M_i \models Env_i \rightarrow (taken(M_i) \Rightarrow \bigwedge_{j \neq i} \mathcal{E}_j)$
P3.	$\models \mathcal{E} \rightarrow \bigwedge_i \mathcal{E}_i$
P4.	$\models (\bigwedge_i \varphi_i) \rightarrow \varphi$
	$M \models Env \rightarrow \varphi$

Example: The critical safety property for Fischer's mutual exclusion algorithm is expressed:

$$Env \rightarrow \Box \neg (at\_\ell_7 \wedge at\_m_7)$$

where the environment restriction Env states that the environment does not modify x. Rule `PAR` may be applied, choosing Env_1 as before, Env_2 chosen symmetrically, and:

$$\varphi_1 : \quad at\_\ell_{7,8} \Rightarrow x = 1$$
$$\varphi_2 : \quad at\_m_{7,8} \Rightarrow x = 2$$

The first premise P1 has been established previously for M_1, and can be established symmetrically for M_2. The third and fourth premises clearly hold. To prove premise P2 for M_2, i.e., that the transitions of M_2 do not violate the environment restriction of M_1, it suffices to consider only the transitions of M_2 that modify x.

For transition τ_{m_4}, first observe that

$$at\_m_4 \Rightarrow \diamondminus_{\leq U}\ x = 0$$

is valid for M_2. It follows that whenever τ_{m_4} is taken, the antecedent $\boxminus_{\leq U}\ x = 1$ of $\mathcal{E}_1^2$ must be false. Furthermore, τ_{m_4} sets $x = 2$, so $\mathcal{E}_1^1$ is also satisfied. Therefore $\mathcal{E}_1 = \mathcal{E}_1^1 \wedge \mathcal{E}_1^2$ holds whenever τ_{m_4} is taken.

For transition τ_{m_8}, which sets $x = 0$, observe that $at\_m_8 \Rightarrow x = 2$ follows from φ_2, so whenever τ_{m_4} is taken, the antecedent $\boxminus_{\leq U}\ x = 1$ of $\mathcal{E}_1^2$ must be false. Again, $\mathcal{E}_1$ is satisfied whenever τ_{m_8} is taken, establishing premise P2 for M_2. The corresponding condition for M_1, stating that the transitions of M_1 satisfy the environment restriction of M_2, can be established symmetrically. ▪

5.3.4 Bounded Response for Fischer's Algorithm

The following bounded response property states that process M_1 is assured access to its critical section in $12 \cdot U$ time units:

$$(at\_\ell_2 \wedge x \neq 1) \Rightarrow \diamond_{\leq 12U}\ at\_\ell_7$$

The environment restriction of M_1, where τ_{E_1} is the environment transition of M_1, is given by:

$$taken(\tau_{E_1}) \Rightarrow \mathcal{E}_1^1 \wedge \mathcal{E}_1^2 \wedge \mathcal{E}_1^3 \wedge \mathcal{E}_1^4 \wedge \mathcal{E}_1^5$$

where each conjunct $\mathcal{E}_1^i$ is described below:

- $\mathcal{E}_1^1: \quad x' = 0 \vee x' = 2 \vee x' = x$

  states that the environment can change x only to 0 or 2.

- $\mathcal{E}_1^2: \quad \boxminus_{\leq U}\ x = 1 \rightarrow x' = 1$

  states that, if $x = 1$ for at least U time units, then the environment does not change x from 1. This implies that M_1 has the right to enter its critical section.

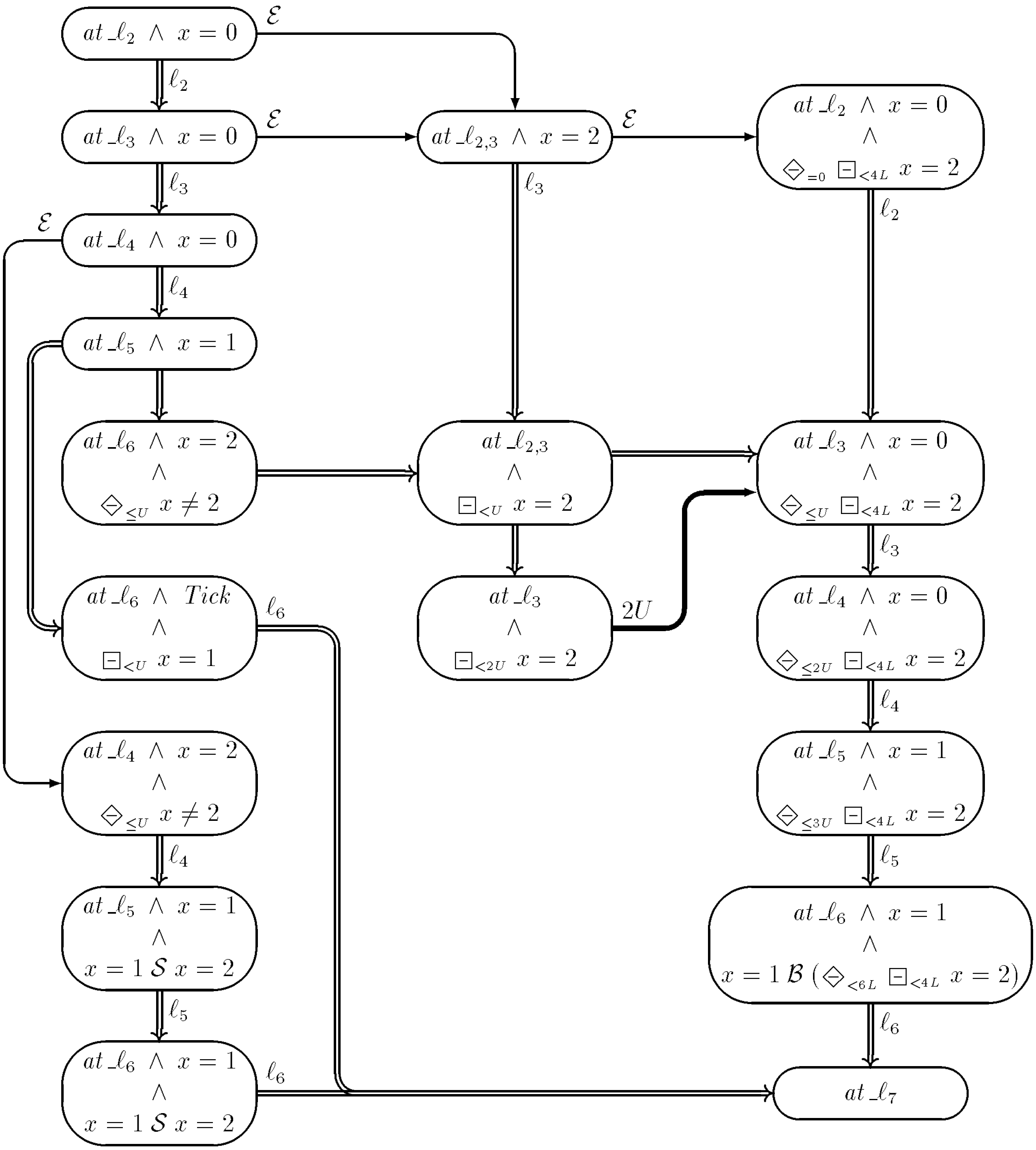

Figure 5.2: A verification diagram.

- $\mathcal{E}_1^3: \quad x = 2 \ \wedge \ \diamondminus_{<4L} \ x \neq 2 \ \rightarrow \ x' = 2$

  states that the environment does not prematurely withdraw a request for its critical section, i.e., cannot change x from 2 in less than $4 \cdot L$ time units.

- $\mathcal{E}_1^4: \quad x = 1 \ \wedge (x = 1) \, \mathcal{S} \, (x = 2) \ \rightarrow \ x' = 1$

  states that, if the environment has lost the competition to enter its critical section, characterized by the change from $x = 2$ to $x = 1$, then the environment cannot attempt to enter the critical section again (until x is reset by M_1).

- $\mathcal{E}_1^5: \quad (x = 1) \, \mathcal{B} \, (\diamondminus_{<6L} \ \boxminus_{<4L} \ x = 2) \ \rightarrow \ x' = x$

  states that the environment cannot attempt to reenter its critical section if M_1 attempts to enter its critical section within $6 \cdot L$ time units after the environment last left, as characterized by $x = 2$ having held for at least $4 \cdot L$ time units.

Applying rule PAR, a suitable specification for M_1 is given as follows:

$$\varphi_1 : \left[\begin{array}{l} at\_\ell_2 \ \wedge \ x \neq 1 \ \Rightarrow \ \left(\begin{array}{l} \diamond_{\leq 7U} \ at\_\ell_7 \\ \quad \vee \ \diamond_{\leq 6U} \ (at\_\ell_3 \ \wedge \ \boxminus_{<2U} \ x = 2) \\ \quad \vee \ \diamond_{\leq 2U} \ (at\_\ell_3 \ \wedge \ x = 2) \end{array} \right) \\ \quad \wedge \\ (at\_\ell_3 \ \wedge \ x = 0 \ \wedge \ \diamondminus_{\leq U} \ \boxminus_{<4L} \ x = 2) \ \Rightarrow \ \diamond_{\leq 4U} \ at\_\ell_7 \\ \quad \wedge \\ (at\_\ell_3 \ \wedge \ x = 2) \ \Rightarrow \ at\_\ell_3 \, \mathcal{W} \, (at\_\ell_3 \ \wedge \ x = 0 \ \wedge \ \diamondminus_{\leq U} \ \boxminus_{<4L} \ x = 2) \end{array} \right]$$

A suitable specification for M_2 is then:

$$\varphi_2 : \left(\begin{array}{l} x = 2 \ \Rightarrow \ \diamond_{\leq 4U} \ (x = 0 \ \wedge \ \diamondminus_{\leq U} \ \boxminus_{<4L} \ x = 2) \\ \quad \wedge \\ \boxminus_{<2U} \ x = 2 \ \Rightarrow \ \diamond_{\leq 2U} \ (x = 0 \ \wedge \ \diamondminus_{\leq U} \ \boxminus_{<4L} \ x = 2) \end{array} \right)$$

A more detailed presentation of this proof appears in Figure 5.2. Let φ be a node in the diagram, connected by thin single or double edges to nodes ψ_i. Then

$$M_1 \ \models \ Env \rightarrow (\varphi \Rightarrow \diamond_{\leq U} \ \bigvee_i \psi_i)$$

is claimed to be valid, and may be established by rule BRESP. For instance

$$at\_\ell_5 \ \wedge \ x = 1 \ \Rightarrow \ \Diamond_{\leq U} \left(\begin{array}{l} at\_\ell_6 \ \wedge \ Tick \ \wedge \ \boxminus_{<U} \ x = 1 \\ \quad \vee \\ at\_\ell_6 \ \wedge \ x = 2 \ \wedge \ \diamondminus_{\leq U} \ x \neq 2 \end{array} \right)$$

may be proven by taking φ_i in rule BRESP to be:

$$\varphi_{U-i} : \quad \diamondminus_{\leq i} \ x = 1 \ \wedge \left(\begin{array}{l} at\_\ell_5 \ \mathcal{S}_{=i} \ at\_\ell_4 \\ \quad \vee \\ at\_\ell_6 \ \wedge \ \boxminus_{<i} \ at\_\ell_{5,6} \ \wedge \ \diamondminus_{\leq i-L} \ at\_\ell_5 \end{array} \right)$$

The thick edges in the diagram correspond to the specification for M_2.

Chapter 6

Conclusions

There are a number of questions that have yet to be resolved. For instance, it is not yet known whether the standard formulas classification, presented in Chapter 2, is complete for the future fragment of temporal logic. In other words, given a κ-formula φ, where κ is one of the temporal property classes in the safety-progress classification, is there a standard future κ-formula equivalent to φ? There is also work to be done in finding a real-time analogue to the safety-progress classification.

The problem of finding suitable component specifications, given a specification for the entire system, also deserves careful study. An alternative, complementary approach to computer-assisted verification is to apply composition rules to manually reduce the verification of a large complex system to the verification of its components, which would then hopefully be amenable to model-checking or other automatic techniques. [KL93] describes one effort in this direction.

Finally, it remains to be seen whether compositional verification methods can be incorporated into an effective, systematic strategy for development.

Bibliography

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In M.S. Paterson, editor, *Proc. 17th Int. Colloq. Aut. Lang. Prog.*, number 443 in Lec. Notes in Comp. Sci, pages 322–335. Springer-Verlag, 1990.

[AFdR80] K.R. Apt, N. Francez, and W.-P. de Roever. A proof system for communicating sequential processes. *ACM Trans. Prog. Lang. Sys.*, 2(3):359–385, 1980.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proc. 10th ACM Symp. Princ. of Dist. Comp.*, pages 139–152, 1991.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. 5th IEEE Symp. Logic in Comp. Sci.*, pages 390–401, 1990.

[AH92] R. Alur and T.A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *Proc. 33rd IEEE Symp. Found. of Comp. Sci.*, pages 177–186, 1992.

[AL89] M. Abadi and L. Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems*, number 430 in Lec. Notes in Comp. Sci, pages 1–41. Springer-Verlag, May/June 1989.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Info. Proc. Lett.*, 82(2):253–284, 1991.

[AL93] M. Abadi and L. Lamport. Conjoining specifications. Technical report, DEC Systems Research Center, October 1993.

[AP93] M. Abadi and G. Plotkin. A logical view of composition and refinement. *Theor. Comp. Sci.*, 114(1):3–30, June 1993.

[Bar86] Howard Barringer. Using temporal logic in the compositional specification of concurrent systems. Technical Report UMCS-86-10-1, Dept. of Computer Science, University of Manchester, October 1986.

[BC88] C. Berthet and E. Cerny. An algebraic model for asynchronous circuits verification. *IEEE Trans. Comp.*, 37(7):835–847, July 1988.

[BK84] Howard Barringer and Ruurd Kuiper. Hierarchical development of concurrent systems in a temporal logic framework. In Stephen D. Brookes et al., editors, *Seminar on Concurrency*, number 197 in Lec. Notes in Comp. Sci. Springer-Verlag, 1984.

[BK85] Howard Barringer and Ruurd Kuiper. Toward the hierarchical, temporal logic, specification of concurrent systems. In *The Analysis of Concurrent Systems*, number 207 in Lec. Notes in Comp. Sci, pages 157–183, 1985.

[BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proc. 16th ACM Symp. Theory of Comp.*, pages 51–63, 1984.

[BKP85] Howard Barringer, Ruurd Kuiper, and Amir Pnueli. A compositional temporal approach to a CSP-like language. In Erich J. Neuhold and Gerhard Chroust, editors, *Formal Models in Programming*. Elsevier Science Publishers B.V., 1985.

[CHMP81] A. Chandra, J. Halpern, A. Meyer, and R. Parikh. Equations between regular terms and an application to process logic. In *Proc. 13th ACM Symp. Theory of Comp.*, pages 384–390, 1981.

[CM88] K.M. Chandy and J. Misra. *Parallel Program Design.* Addison-Wesley, 1988.

[CMP92] E. Chang, Z. Manna, and A. Pnueli. The safety-progress classification. In H. Schwichtenberg, editor, *Logic, Algebra, and Computation*, NATO ASI Series, Subseries F: Computer and System Sciences. Springer-Verlag, 1992.

[Col93] P. Collette. Application of the composition principle to unity-like specifications. In *TAPSOFT'93: Theory and Practice of Software Development*, number 668 in Lec. Notes in Comp. Sci, pages 230–242. Springer-Verlag, 1993.

[dR85] W.-P. de Roever. The quest for compositionality. In E.J. Neuhold and G. Chroust, editors, *Formal Models in Programming.* Elsevier Science Publishers (North-Holland), 1985.

[FFG91] L. Fix, N. Francez, and O. Grumberg. Program composition and modular verification. In J. Leach, B. Monien, and M. Rodríguez Artalejo, editors, *Proc. 18th Int. Colloq. Aut. Lang. Prog.*, number 510 in Lec. Notes in Comp. Sci, pages 93–114. Springer-Verlag, 1991.

[GU84] R. Gerth and U. Utrecht. Transition logic. In *Proc. 16th ACM Symp. Theory of Comp.*, pages 39–50, 1984.

[HKP80] D. Harel, D. Kozen, and R. Parikh. Process logic: Expressiveness, decidability, and completeness. In *Proc. 21st IEEE Symp. Found. of Comp. Sci.*, pages 129–142, 1980. Also in *J. Comp. Sys. Sci.* 25 (1982), 144–170.

[HMM83] J. Halpern, Z. Manna, and B. Moszkowski. A hardware semantics based on temporal intervals. In *Proc. 10th Int. Colloq. Aut. Lang. Prog.*, number 154 in Lec. Notes in Comp. Sci, pages 278–291. Springer-Verlag, 1983.

[HMP93] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Inf. and Cont.*, 1993.

[HO83] Brent Hailpern and Susan Owicki. Modular verification of computer communication protocols. *IEEE Trans. on Communications*, 31(1):56–68, January 1983.

[HP85] D. Harel and D. Peleg. Process logic with regular formulas. *Theor. Comp. Sci.*, 38:307–322, 1985.

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In *Proceedings of the IFIP 9th World Congress*, pages 321–332, 1983.

[Jon87] B. Jonsson. Modular verification of asynchronous networks. In *Proc. 6th ACM Symp. Princ. of Dist. Comp.*, pages 137–151, 1987.

[Jon89] B. Jonsson. On decomposing and refining specifications of distributed systems. In *Stepwise Refinement of Distributed Systems*, number 430 in Lec. Notes in Comp. Sci, pages 361–385. Springer-Verlag, May/June 1989.

[JP87] M. Joseph and P.K. Pandya. Specification and verification of total correctness of distributed programs. Technical Report 96, Dept. of Computer Science, University of Warwick, February 1987.

[KL93] R. Kurshan and L. Lamport. Verification of a multiplier: 64 bits and beyond. In C. Courcoubetis, editor, *Proc. 5th Int. Conf. CAV*, number 697 in Lec. Notes in Comp. Sci, pages 166–179. Springer-Verlag, 1993.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.

[Lam80] L. Lamport. The "Hoare logic" of concurrent programs. *Acta Informatica*, 14:21–37, 1980.

[Lam83] L. Lamport. Specifying concurrent program modules. *ACM Trans. Prog. Lang. Sys.*, 5(2), April 1983.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. of the Workshop on Logics of Programs*, number 193 in Lec. Notes in Comp. Sci, pages 196–218. Springer-Verlag, 1985.

[LS83] L. Lamport and F.B. Schneider. The "Hoare logic" of CSP, and all that. *ACM Trans. Prog. Lang. Sys.*, 6(2):281–296, 1983.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. Princ. of Dist. Comp.*, pages 137–151, 1987.

[MC81] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Engin.*, 7(4):417–426, July 1981.

[MCS82] J. Misra, K.M. Chandy, and T. Smith. Proving safety and liveness of communicating processes with examples. In *Proc. ACM Symp. Princ. of Dist. Comp.*, pages 201–208, 1982.

[MP91] Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[MP93] Z. Manna and A. Pnueli. Temporal specification and verification of reactive modules. *J. ACM*, 1993. To appear.

[NDGO85] V. Nguyen, A. Demers, D. Gries, and S. Owicki. Behavior: a temporal approach to process modeling. In *Proc. of the Workshop on Logics of Programs*, number 193 in Lec. Notes in Comp. Sci, pages 237–254. Springer-Verlag, 1985.

[NGO85] V. Nguyen, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. In *Proc. 12th ACM Symp. Princ. of Prog. Lang.*, pages 121–131, January 1985.

[OG76a] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs i. *Acta Informatica*, 6:319–340, 1976.

[OG76b] S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Comm. ACM*, 19(5):279–285, 1976.

[PJ91] P.K. Pandya and M. Joseph. P-A logic — a compositional proof system for distributed programs. *Dist. Comp.*, 5:37–54, 1991.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Found. of Comp. Sci.*, pages 46–57, 1977.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K.R. Apt, editor, *Logics and Models of Concurrent Systems*, NATO ASI F13, pages 123–144. Springer-Verlag, 1985.

[RP86] R. Rosner and A. Pnueli. A choppy logic. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 306–313, 1986.

[SC82] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. In *Proc. 14th ACM Symp. Theory of Comp.*, pages 159–168, 1982.

[Sta85] E.W. Stark. A proof technique for rely/guarantee properties. In S.N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, number 206 in Lec. Notes in Comp. Sci, pages 369–391. Springer-Verlag, 1985.

[Sto74] L.J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, Massachusetts Institute of Technology, 1974.

[Wol81] P. Wolper. Temporal logic can be more expressive. In *Proc. 22nd IEEE Symp. Found. of Comp. Sci.*, pages 340–348, 1981.

[ZdBdR84] J. Zwiers, A. de Bruin, and W.-P. de Roever. A proof system for partial correctness of dynamic networks of processes. In *Proceedings of the Conference on Logics of Programs 1983*, number 164 in Lec. Notes in Comp. Sci, 1984.

[ZdR89] J. Zwiers and W.-P. de Roever. Compositionality and modularity in process specification and design: A trace-state based approach. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, number 398 in Lec. Notes in Comp. Sci, pages 351–374. Springer-Verlag, April 1989.

[Zwi89] J. Zwiers. *Compositionality, Concurrency, and Partial Correctness.* Number 321 in Lec. Notes in Comp. Sci. Springer-Verlag, 1989.