Chapter 24

Model Checking

Edmund M. Clarke

Bernd-Holger Schlingloff

Second readers: Nikolaj Bjorner and Perdita Stevens.

Contents

1	Introduction	1637
2	Logical Languages, Expressiveness	1641
	2.1 Propositional and First Order Logic	1642
	2.2 Multimodal and Temporal Logic	1644
	2.3 Expressive Completeness of Temporal Logic	1648
3	Second Order Languages	1654
	3.1 Linear and Branching Time Logics	1654
	3.2 Propositionally Quantified Logics	1656
	3.3 ω-automata and ω-languages	1665
	3.4 Automata and Logics	1668
4	Model Transformations and Properties	1670
	4.1 Models, Automata and Transition Systems	1671
	4.2 Safety and Liveness Properties	1673
	4.3 Simulation Relations	1676
5	Equivalence reductions	1681
	5.1 Bisimulations (p-morphisms)	1682
	5.2 Distinguishing Power and Ehrenfeucht-Fraïssé Games	1685
	5.3 Auto-bisimulations and the Paige/Tarjan Algorithm	1687
6	Completeness	1689
	6.1 Deductions in Multimodal Logic	1691
	6.2 Transitive Closure Operators	1695
7	Decision Procedures	1700
	7.1 Deciding Branching Time Logics	1700
	7.2 Satisfiability Algorithms for Natural Models	1704
8	Basic Model Checking Algorithms	1711
	8.1 Global Branching Time Model Checking	1712
	8.2 Local Linear Time Model Checking	1716
	8.3 Model Checking for Propositional μ-Calculus	1720
9	Modelling of Reactive Systems	1724
	9.1 Parallel Programming Paradigms	1724
	9.2 Some Concrete Formalisms for Finite State Systems	1726

HANDBOOK OF AUTOMATED REASONING
Edited by Alan Robinson and Andrei Voronkov
© 2001 Elsevier Science Publishers B.V. All rights reserved

9.3 Example Applications . . . 1727
10 Symbolic Model Checking . . . 1735
10.1 Binary Decision Diagrams . . . 1736
10.2 Symbolic Model Checking for **CTL** . . . 1744
10.3 Relational μ-Calculus . . . 1746
11 Partial Order Techniques . . . 1751
11.1 Stuttering Invariance . . . 1752
11.2 Partial Order Analysis of Elementary Nets . . . 1754
12 Bounded Model Checking . . . 1755
12.1 An Example . . . 1756
12.2 Translation into Propositional Logic . . . 1757
13 Abstractions . . . 1759
13.1 Abstraction functions . . . 1759
13.2 Symmetry Reductions . . . 1762
13.3 Parameterized Systems . . . 1763
14 Compositionality and Modular Verification . . . 1764
14.1 Model Checking and Theorem Proving . . . 1765
14.2 Compositional Assume-Guarantee Reasoning . . . 1766
15 Further Topics . . . 1767
15.1 Combination of Heuristics . . . 1768
15.2 Real Time Systems . . . 1769
15.3 Probabilistic Model Checking . . . 1770
15.4 Model Checking for Security Protocols . . . 1771
Bibliography . . . 1774
Index . . . 1788

1. Introduction

Model checking is an automatic technique for verifying correctness properties of safety-critical reactive systems. This method has been successfully applied to find subtle errors in complex industrial designs such as sequential circuits, communication protocols and digital controllers [Browne, Clarke and Dill 1985, Clarke, Emerson and Sistla 1986, Clarke, Long and McMillan 1991, Burch, Clarke, Dill, Long and McMillan 1994]. It is expected that besides classical quality assurance measures such as static analysis and testing, model checking will become a standard procedure in the design of reactive systems.

A *reactive system* [Harel and Pnueli 1985, Manna and Pnueli 1992, Manna and Pnueli 1995] consists of several components which are designed to interact with one another and with the system's environment. In contrast to *functional* (or *transformational*) systems, in which the semantics is given as a function from input to output values, a reactive system is specified by its temporal properties. A (temporal) *property* is a set of desired behaviors in time; the system satisfies the property if each execution of the system belongs to this set. From a logical viewpoint, the system is described by a semantical (Kripke-)*model*, and a property is described by a logical *formula*. Arguing about system correctness, therefore, amounts to determining the *truth of formulas in models*.

In order to be able to perform such a verification, one needs a *modelling language* in which the system can be described, a *specification language* for the formulation of properties, and a deductive *calculus* or *algorithm* for the verification process. Usually, the system to be verified is modeled as a (finite) state transition graph, and the properties are formulated in an appropriate propositional temporal logic. An efficient search procedure is then used to determine whether or not the state transition graph satisfies the temporal formulas. When model checking was first developed in 1981 [Clarke and Emerson 1981, Emerson and Clarke 1982, Quielle and Sifakis 1981], it was only possible to handle concurrent systems with a few thousand states. In the last few years, however, the size of the concurrent systems that can be handled has increased dramatically. By using sophisticated data structures and heuristic search procedures, it is now possible to check systems many orders of magnitude larger [Burch, Clarke, McMillan, Dill and Hwang 1992].

Much of the success of model checking is due to the fact that it is a fully automatic verification method. Interactive methods are more general but harder to use; automatic methods have a limited range but are more likely to be accepted. In interactive verification, the user provides the overall proof strategy; the machine augments this by

- checking the correctness of each step,
- maintaining a list of assumptions and subgoals,
- applying the rules and substitutions which the user indicates, and by
- searching for applicable transformation rules and assumptions.

Sophisticated tools are also able to prove certain lemmas automatically, usually by applying a heuristic search. Although there has been considerable research on the use of theorem provers, term rewriting systems and proof checkers for verification, these techniques are time consuming and often require a great deal of manual intervention. Moreover, since most interactive provers are designed for undecidable languages (e.g., first or higher order logic), the proof process can never be completely automatic. User interaction is required, e.g., to find loop invariants or inductive hypotheses, and only an experienced user can perform a nontrivial proof.

On the other hand, with model checking all the user has to provide is a model of the system and a formulation of the property to be proven. The verification tool will either terminate with an answer indicating that the model satisfies the formula or show why the formula fails to hold in the model. These counterexamples are particularly helpful in locating errors in the model or system.

With the completely automatic approach it may be necessary for the model checking algorithm to traverse all reachable states of the system. This is only possible if the state space is finite. Whereas other automated deduction methods may be able to handle some infinite-state problems, model checking usually is constrained to a finite abstraction. In fact, model checking algorithms can be regarded as decision procedures for temporal properties of finite-state reactive systems. However, many interesting systems like sequential circuits or network protocols are finite state. Moreover, in the design of safety critical systems it is often possible to separate the (finite state) control structure from the (infinite state) data structure of a given module. Finally, in many cases it is possible to *abstract* an infinite domain into an appropriate finite one, such that "interesting" properties are preserved. In an 'a posteriori' verification, some efforts may be necessary to construct such an abstraction from a given program. In a structured software development process, however, the abstract system often arises naturally during an early design phase.

A main impediment of the fully automatic approach is the state explosion: if any state of the system is uniquely described by n state bits, then there are 2^n possible states the system can be in. At the present time, the number of states that can be represented *explicitly* (e.g., by lists or hash tables) is approximately 10^6. In [Burch, Clarke, McMillan, Dill and Hwang 1992, McMillan 1993], *binary decision diagrams* (BDDs) were used to represent state spaces *symbolically*. With this technique, models with several hundred state bits and more than 10^{100} reachable states can be checked. Because of this and other technical advances in the available tools it is now possible to verify reactive systems of realistic industrial complexity, and a number of major companies including Intel, Motorola, ATT, Fujitsu and Siemens have started using symbolic model checkers to verify actual designs.

We now describe a concrete example of a nontrivial application, where model checking has been used to improve a proposed international standard. Consider the cache coherence protocol described in the draft IEEE Futurebus+ standard [IEEE 1994]. This protocol is required to insure *coherence*: consistency of data in hierarchical systems composed of many processors and caches interconnected by multiple bus segments. Such protocols are notoriously complex and, therefore, quite difficult to debug. The Futurebus+ protocol maintains coherence by having the individual caches observe all bus transactions. In order to increase performance, the protocol allows transactions to be *split*. That is, the completion of a transaction may be delayed and the bus freed. Then, it is possible to service local requests while the remote request is being processed. At some later time, an explicit response is issued to complete the transaction. Consider a sample configuration with two processors P_1 and P_2 accessing data from a common memory via a single bus (see Fig. 1 on page 1640). Initially, neither processor has a copy of the data in its cache; they are said to be in the `invalid` state. Processor P_1 issues a `read_shared` request to obtain a readable copy of the data from memory. P_2 may observe this transaction and also obtain a readable copy, such that at the end of the transaction, both caches contain a `shared_unmodified` copy of the data. Next, if P_1 decides to modify the data, the copy held by P_2 must be eliminated in order to maintain coherence. Therefore, P_1 issues an `invalidate` transaction on the bus. When P_2 notices this transaction, it purges the data from its cache. After executing the `invalidate`-transaction, P_1 now has an `exclusive` copy of the data.

The standard specifies the possible states of the cache data within each processor and how this state is updated during each possible transaction. It consists of roughly 300 so-called *attributes*, which are essentially boolean variables together with some rules for setting and clearing them. In the automated verification of the Futurebus+ protocol described in [Clarke, Grumberg, Hiraishi, Jha, Long, McMillan and Ness 1993], these attributes were transformed into the input language of the SMV model checker [McMillan 1993]. For example, the following SMV code fragment indicates how the cache state is updated when the cache issues a `read_shared` transaction:

```
next(state) :=
  case CMD=read_shared:
    case state=invalid:
      case !SR & !TF: exclusive_unmodified;
           !SR      : shared_unmodified;
           1        : invalid;
      esac;
    ...
    esac;
  ...
  esac;
```

If the transaction is not split (`!SR`), then the data will be supplied to the cache. Either no other caches will read the data (`!TF`), in which case the cache obtains an `exclusive_unmodified` copy, or some other cache also obtains the data, and everyone obtains `shared_unmodified` copies. If the transaction is split, the cache data remains in the invalid state.

The model for the cache coherence protocol consists of approximately 2300 lines of SMV code (not counting comments). The model is highly nondeterministic, both to reduce the complexity of verification by hiding details, and to cover allowed design choices. This model is compiled into an internal BDD representation by the SMV program. Correctness properties are formulated in the temporal logic **CTL**. For example, cache consistency is described by requiring that if two caches have copies of a cache line, then they agree on the data in that line:

```
AG (P1.readable & P2.readable -> P1.data = P2.data)
```

This formula is evaluated automatically on the BDD representation of the model. SMV finds that it is not valid and exhibits a scenario which could lead to the error: initially, both caches are invalid. Processor P_1 obtains an `exclusive_unmodified` copy of the data (say, `data1`) as described above and the data of P_2 is `invalid` (see Fig. 1). Then, P_2 issues a `read_modified`, which P_1 splits for invalidation. That is, the memory supplies a copy of the data to P_2, and P_1 postpones the invalidation of cache data until local actions are completed. Still having an `exclusive_unmodified` copy of `data1`, P_1 now modifies the data (say, into `data2`) and transitions to `exclusive_modified`. At this point, P_1 and P_2 are inconsistent. This bug can be fixed by requiring P_1 to go to the `shared_unmodified` state when it splits the `read_modified` transaction for invalidation.

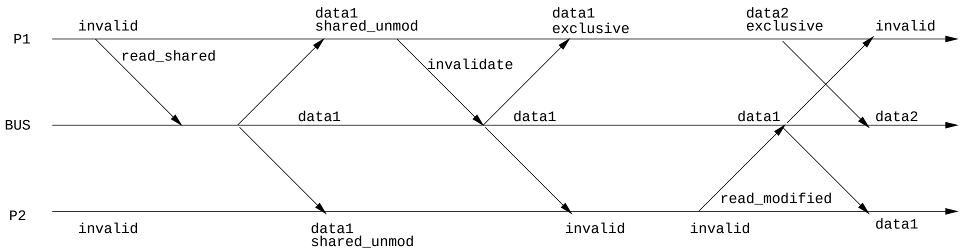


Figure 1: Error scenario in the Futurebus+ protocol

Given a formal model of a system to be verified, and a formulation of the properties the system should satisfy, there are three possible results which an automated model checker can produce:

1. either it finds a *proof* for the formula in the model and outputs "verified", or
2. it constructs a *refutation*, i.e., an execution of the (model of the) system which dissatisfies the (formulation of the) property, or
3. the complexity of the verification procedure exceeds the given memory limit or time bound.

If there is not sufficient space or time, in some cases it is possible to use bigger and faster machines for verification. Alternatively, one can use a coarser abstraction of the system and its properties. The third possibility is to employ heuristics which improve the performance of the verifier. Some of these heuristics are discussed in Sections 10 and 11.

In some sense it is more interesting to get a refutation than to get a proof. With a refutation, one can decide whether it is due to the modelling and formulation, or whether this undesired sequence of events could indeed happen in reality. In the former case, the unrealistic behavior can be eliminated by additional assumptions on the model or formula. In the latter case, one has found a bug, and the system and model can be changed appropriately. One of the major advantages of the fully automatic approach is that there is almost no additional overhead for the new verification of the changed system.

If the model checker is able to prove all specified formulas for the given model, then the verification is successfully completed. However, there can never be any guarantee that a system which has been verified by a computer tool will function correctly in reality. Even if we could assume that the verifier's hard- and software is correct (which we can not), there is a fundamental source of inaccuracy involved. Verification proves theorems about models of systems and formulations of properties, not about physical systems and desired behavior; we can never know to what extent our models and formulations reflect physical reality and intuitions. It is not possible to guarantee that a physical system will behave correctly in unexpected (i.e., unmodeled) situations. It would be unreasonable, however, to reject formal methods because they cannot offer such guarantees. Civil engineering can never *prove* that a certain building will not collapse. Nevertheless it uses mathematical models to calculate loads and wall thicknesses and so on. Similarly, we can never *prove* that our model adequately represents the reality. Therefore we can never *prove* that a system will function as planned. Nevertheless, compared to current practice, the use of formal methods can significantly decrease the amount of errors in complex software systems. A temporal logic specification adds redundancy to the design by restating an intended property in a (different) concise formalism. Computer aided verification can help to *locate errors* and to *increase reliability* of these systems. In the future, formal verification by model checking will augment classical software design tools such as structured analysis, code review and testing.

In this survey, we give a tutorial on the theoretical foundations and techniques used in model checking. Starting with elementary material on propositional temporal logics and automata we derive basic model checking algorithms from completeness results and tableau decision procedures. Then we discuss applications and techniques for efficient implementation of these algorithms. We extend the results to more expressive logics and models. Finally, we discuss some open problems and future research directions in the area. At the end of this chapter, the reader can find a list of all symbols and notations and an index of topics.

2. Logical Languages, Expressiveness

One of the major concerns of philosophical logic is to find an appropriate language for the formalization of natural language reasoning. The first and probably most successful of these languages is first order logic. Almost all mathematical statements and proofs can be formulated in this language. However, certain concepts important for computer science like well-foundedness and transitive closure require more expressive languages.

Temporal logic was invented to formalize natural language sentences about events in time, which use temporal adverbs like "eventually" and "constantly". Temporal logics have proved to be useful for specifying concurrent systems, because they can describe the ordering of events without introducing time explicitly. There have been many variants of temporal logic proposed in the literature. Temporal logics can be classified as

- state- or transition- (interval-) based, depending on whether the formulated properties involve one or more reference points,
- linear or branching time, depending on the intuition of time as a sequence or as a tree of events,
- star-free or regular, depending on the formal languages which can be defined by formulas of the logic, and
- propositional or first-order, depending on the cardinality of the nontemporal domains.

In principle, these classifications are orthogonal; in practice, however, only certain combinations are widely used. In this survey, we concentrate on propositional modal logic, linear temporal logic, computation tree logic, and fixpoint calculus. Restrictions and extensions of these logics are introduced whenever appropriate.

2.1. Propositional and First Order Logic

We assume a set $\mathcal{P} = \{\mathtt{p}, \mathtt{q}, \mathtt{p}_1, ...\}$ of (atomic) propositions which can be either true or false. [1] For example, the proposition `stack_is_empty` denotes the fact that "the stack is empty". The *propositional logic* **PL** is built from $\mathcal{P}$ with the following syntax:

$$\mathbf{PL} \quad ::= \quad \mathcal{P} \quad | \quad \bot \quad | \quad (\mathbf{PL} \rightarrow \mathbf{PL})$$

That is,

- Every $\mathtt{p} \in \mathcal{P}$ is a well-formed formula of propositional logic,
- $\bot$ is a well-formed formula ("the falsum"),
- if φ and ψ are well-formed formulae, then so is $(\varphi \rightarrow \psi)$, and
- nothing else is a formula.

$\mathcal{P}$ is a parameter of the logic; the special case $\mathcal{P} = \{\}$ is allowed. Other connectives can be defined as usual: $\neg\varphi \triangleq (\varphi \rightarrow \bot)$, $\top \triangleq \neg\bot$, $(\varphi \vee \psi) \triangleq (\neg\varphi \rightarrow \psi)$, $(\varphi \wedge \psi) \triangleq \neg(\neg\varphi \vee \neg\psi)$, and $(\varphi \leftrightarrow \psi) \triangleq ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi))$. The precedence of these operators is fixed by $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$, and parentheses are omitted in formulas whenever appropriate. Atomic propositions and negated propositions are called *literals*.

An *interpretation* $\mathcal{I}$ for the propositions is a function assigning a truth value from $\{\mathbf{true}, \mathbf{false}\}$ to every proposition. (For example, the proposition `stack_is_empty` is interpreted differently on a farm, in a library, or in front of a computer terminal.) A *propositional model* $\mathcal{M} \triangleq (U, \mathcal{I})$ consists of the fixed binary domain $U \triangleq \{\mathbf{true}, \mathbf{false}\}$ and an interpretation for $\mathcal{P}$. (Later on, we will consider logics

[1] A list of syntactic categories and other symbols is given in the appendix.

over arbitrary nonbinary domains.) The most basic semantical notion is the *validation relation* $\models$ between a model $\mathcal{M}$ and a formula φ. It is defined by the following clauses.

- $\mathcal{M} \models \mathtt{p}$ iff $\mathcal{I}(\mathtt{p}) = \mathbf{true}$,
- $\mathcal{M} \not\models \bot$, and
- $\mathcal{M} \models (\varphi \to \psi)$ iff $\mathcal{M} \models \varphi$ implies $\mathcal{M} \models \psi$.

That is, $\mathcal{M} \models (\varphi \to \psi)$ iff $\mathcal{M} \not\models \varphi$ or $\mathcal{M} \models \psi$. If $\mathcal{M} \models \varphi$, then we say that $\mathcal{M}$ *validates* φ, or, equivalently, φ is *valid in* $\mathcal{M}$.

Propositional logic is not well-suited to formalize statements about events in time. Even though the interpretation of a statement can be fixed, its truth value may vary in time. This cannot be expressed directly in **PL**.

To express such temporal dependencies, first order logic can be used. The set $\mathcal{P}$ is redefined to be a set of *monadic predicates*. That is, each $\mathtt{p} \in \mathcal{P}$ is augmented with an additional parameter denoting time, for example, $\mathtt{stack\_is\_empty}(t)$.

For sake of simplicity, we do not include function symbols (or constants) in the first-order language. Assume in addition to the set $\mathcal{P}$ of unary predicates a fixed set $\mathcal{R} \triangleq \{R, a, b, ...\}$ of *accessibility relations*, and let $\mathcal{R}^+ \triangleq \mathcal{R} \cup \{\prec, <, =\}$. Furthermore, let $\mathcal{T}$ be a set of *first-order variables* $\mathcal{T} \triangleq \{t, t_0, ...\}$ for points in time (which is assumed to be infinite unless stated otherwise).

$$\mathbf{FOL} \quad ::= \quad \mathcal{P}(\mathcal{T}) \mid \bot \mid (\mathbf{FOL} \to \mathbf{FOL}) \mid \mathcal{R}^+(\mathcal{T}, \mathcal{T}) \mid \exists \mathcal{T}\ \mathbf{FOL}$$

When writing formulas, we often use infix notation for relational terms: $t_1 R t_2 \triangleq R(t_1, t_2)$. The notation $\forall t\ \varphi$ is an abbreviation for $\neg\exists t\ \neg\varphi$, the string $x > y$ stands for $y < x$, and $x \leq y$ for $(x < y \vee x = y)$, etc.

To assign a truth value to a formula containing (free) variables, we assume that we are given a nonempty *universe* U of *points* in time, and that the *interpretation* $\mathcal{I}$ assigns to every proposition $\mathtt{p} \in \mathcal{P}$ a subset of points $\mathcal{I}(\mathtt{p}) \subseteq U$, and to every relation symbol $R \in \mathcal{R}$ a binary relation $\mathcal{I}(R) \subseteq U \times U$. For the special relation signs $=$, $\prec$, and $<$ we require that $\mathcal{I}(=) \triangleq \{(w, w) \mid w \in U\}$ is the *equality relation*, $\mathcal{I}(\prec) \triangleq \bigcup\{\mathcal{I}(R) \mid R \in \mathcal{R}\}$ is the *transition relation*, and $\mathcal{I}(<)$ is the transitive closure of $\mathcal{I}(\prec)$, the *reachability relation*. A *variable valuation* $\mathbf{v}$ assigns to any variable $t \in \mathcal{T}$ a point $w \in U$. A first-order model $\mathcal{M} \triangleq (U, \mathcal{I}, \mathbf{v})$ consists of a universe U, an interpretation $\mathcal{I}$, and a variable valuation $\mathbf{v}$. As in the propositional case, we define when a formula holds in a model:

- $\mathcal{M} \models \mathtt{p}(t)$ iff $\mathbf{v}(t) \in \mathcal{I}(\mathtt{p})$;
- $\mathcal{M} \not\models \bot$, and
- $\mathcal{M} \models (\varphi \to \psi)$ iff $\mathcal{M} \models \varphi$ implies $\mathcal{M} \models \psi$;
- $\mathcal{M} \models R(t_0, t_1)$ iff $(\mathbf{v}(t_0), \mathbf{v}(t_1)) \in \mathcal{I}(R)$;
- $\mathcal{M} \models \exists t\ \varphi$ iff $(U, \mathcal{I}, \mathbf{v}') \models \varphi$ for some $\mathbf{v}'$ which differs from $\mathbf{v}$ at most in t.

This language is rather expressive: consider the following example formulas.

(1) $(\mathtt{stack\_is\_empty}(t_0) \to \exists t_1(put(t_0, t_1) \land \neg\mathtt{stack\_is\_empty}(t_1)))$
If `stack_is_empty`, then it is possible to perform a *put* such that not `stack_is_empty` holds.

(2) $\forall t_1((t_0 \leq t_1 \land \mathtt{req}(t_1)) \to \exists t_2(t_1 < t_2 \land \mathtt{ack}(t_2)))$
Every request is eventually acknowledged.

(3) $\forall t_1((t_0 \leq t_1 \land \mathtt{req}(t_1)) \to \exists t_2((t_1 < t_2 \land \mathtt{ack}(t_2)) \land \forall t_3((t_1 < t_3 \land t_3 < t_2) \to \mathtt{req}(t_3))))$
No request is withdrawn before it is acknowledged.

2.2. Multimodal and Temporal Logic

First order logic has been criticized by theoretical linguists for not being intuitive. Except from text in mathematical books, one can hardly find English sentences which explicitly use variables to refer to objects. Natural language statements use modal adverbs like "possibly" and "necessarily" to refer to an alternative state of affairs. Temporal phrases in natural language use the adverbs "eventually" and "constantly" (or "sometime" and "always") to refer to future points in time. Modal logic was invented to formalize these modal and temporal adverbs [Lewis 1912, Prior 1957, Prior 1967]. The idea is to suppress first-order variables $t \in \mathcal{T}$; propositions $\mathtt{p} \in \mathcal{P}$ are nullary again. In modal logics, the meaning of a proposition like `stack_is_empty` is intended to be "the stack is empty *now*". Thus, in a temporal interpretation, every formula describes a certain state of affairs *at a given point*.

To be able to describe properties depending on the relations between points, in multimodal logic for every $R \in \mathcal{R}$ a new operator $\langle R\rangle\,\varphi$ is introduced. The meaning of $\langle R\rangle\,\varphi$ is "possibly φ", i.e., "there exists some t accessible via R such that φ holds at t". Dually, $[R]\,\varphi \triangleq \neg\langle R\rangle\,\neg\varphi$ means "necessarily φ"; "for all t accessible via R, it is the case that φ holds at t".

$$\mathbf{ML} \ ::= \ \mathcal{P} \ | \ \bot \ | \ (\mathbf{ML} \to \mathbf{ML}) \ | \ \langle\mathcal{R}\rangle\,\mathbf{ML}.$$

Intuitively, the above example (1) could be written

$$(\mathtt{stack\_is\_empty} \to \langle put\rangle\,\neg\mathtt{stack\_is\_empty}).$$

Assume again that U is a nonempty set of *points in time* (or "possible worlds"). An *interpretation* $\mathcal{I}$ for multimodal logic assigns to every $\mathtt{p} \in \mathcal{P}$ and $R \in \mathcal{R}$ a subset $\mathcal{I}(\mathtt{p}) \subseteq U$ and a relation $\mathcal{I}(R) \subseteq U \times U$, respectively. The tuple $\mathcal{F} \triangleq (U, \mathcal{I})$ is called a *frame* for $\mathcal{P}$ and $\mathcal{R}$. A (Kripke-) *model* (introduced in [Kripke 1963, Kripke 1975]) $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$ for multimodal logic consists of a frame $(U, \mathcal{I})$ and a *current point* $w_0 \in U$. If $\mathcal{M} = (U, \mathcal{I}, w_0)$, we say that $\mathcal{M}$ is *based on* the frame $\mathcal{F} = (U, \mathcal{I})$. Thus, a Kripke model for multimodal logic is similar to a first order model, where the variable valuation $\mathbf{v}$ is replaced by a single designated point w_0.

Note that our notion of frame and model is somewhat different from the traditional use of these terms, where a *frame* denotes the tuple $(U, \{\mathcal{I}(R) \mid R \in \mathcal{R}\})$, and a *model* is the triple $(U, \{\mathcal{I}(R) \mid R \in \mathcal{R}\}, \{\mathcal{I}(\mathtt{p}) \mid \mathtt{p} \in \mathcal{P}\})$. Historically, atomic propositions have been regarded as being "variable" in a formula, thus $\{\mathcal{I}(\mathtt{p}) \mid \mathtt{p} \in \mathcal{P}\}$ is a separate *valuation* for these variables. In this paper, a proposition denotes a fixed predicate, hence its meaning is given by the interpretation. In a later section we introduce a separate syntactic category of *proposition variables*, which can be *evaluated* differently in each context.

Validity of a modal formula in a Kripke model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$ is defined as follows.

- $\mathcal{M} \models \mathtt{p}$ iff $w_0 \in \mathcal{I}(\mathtt{p})$;
- $\mathcal{M} \not\models \bot$, and
- $\mathcal{M} \models (\varphi \to \psi)$ iff $\mathcal{M} \models \varphi$ implies $\mathcal{M} \models \psi$.
- $\mathcal{M} \models \langle R \rangle \varphi$ iff there exists $w_1 \in U$ with $(w_0, w_1) \in \mathcal{I}(R)$ and $(U, \mathcal{I}, w_1) \models \varphi$.

We write $w \models \varphi$ instead of $(U, \mathcal{I}, w) \models \varphi$ whenever the frame $(U, \mathcal{I})$ is given. A formula φ is *universally* valid (or *frame-valid*) in $(U, \mathcal{I})$, if for all $w \in U$ it holds that $w \models \varphi$.

As defined above, $\prec$ is interpreted as the *transition relation*, i.e., the union of all accessibility relations, $<$ is interpreted as the transitive closure of $\prec$, and $\leq$ as the reflexive transitive closure (the *reachability relation*). For these special relations $\sim \in \{\prec, <, =, \leq\}$, we henceforth simply write $v \sim w$ instead of $(v, w) \in \mathcal{I}(\sim)$. We introduce the special operators $\mathbf{X}$, $\mathbf{F}^{+}$ and $\mathbf{F}^{*}$:

- $w_0 \models \mathbf{X}\,\varphi$ iff there exists $w_1 \in U$ such that $w_0 \prec w_1$ and $w_1 \models \varphi$,
- $w_0 \models \mathbf{F}^{+}\varphi$ iff there exists $w_1 \in U$ such that $w_0 < w_1$ and $w_1 \models \varphi$, and
- $w_0 \models \mathbf{F}^{*}\varphi$ iff there exists $w_1 \in U$ such that $w_0 \leq w_1$ and $w_1 \models \varphi$.

For the dual operators, we use the symbols $\cancel{\mathbf{X}}\,\varphi \triangleq \neg\,\mathbf{X}\,\neg\varphi$, and $\mathbf{G}^{+}\varphi \triangleq \neg\,\mathbf{F}^{+}\neg\varphi$, and $\mathbf{G}^{*}\varphi \triangleq \neg\,\mathbf{F}^{*}\neg\varphi$. Traditionally, $\mathbf{X}$, $\mathbf{F}$, and $\mathbf{G}$ have been used to indicate ne**X**t time, **F**uture and **G**lobal operators[2]. Alternatively, $\mathbf{F}^{+}$ and $\mathbf{G}^{+}$ are called *sometime*- and *always*-operators. $\cancel{\mathbf{X}}$ is referred to as *weak next*- operator.

Here are some historical remarks on the use of these operators. In the 1950's and 1960's, proof theory and model theory of modal logic was developed ([Rescher and Urquhart 1971, Hughes and Cresswell 1977] are historical, and [Blackburn, de Rijke and Venema 2000] is a modern textbook on this topic). Its applicability to computer science was discovered in the 1970's: [Burstall 1974] suggested a modal logic built upon $\mathbf{F}^{+}$ and $\mathbf{G}^{+}$ to describe program properties. [Kröger 1978] suggested to use both $\mathbf{X}$ and $\mathbf{F}^{+}$ for program verification. [Pnueli 1977] used a similar system for parallel programs. [Gabbay, Pnueli, Shelah and Stavi 1980] extended temporal logic for program specification by the binary connective *until* (explained below). The framework was further elaborated in [Pnueli 1981, Manna and Pnueli 1981, Manna and Pnueli 1982*b*, Manna and Pnueli 1982*a*, Pnueli 1984, Harel and Pnueli 1985,

[2] A note on notation: with the above convention, the $\mathbf{X}$, $\cancel{\mathbf{X}}$, $\mathbf{F}^{+}$, $\mathbf{F}^{*}$, $\mathbf{G}^{+}$ and $\mathbf{G}^{*}$ operators could be written as $\langle\prec\rangle$, $[\prec]$, $\langle<\rangle$, $\langle\leq\rangle$, $[<]$ and $[\leq]$, respectively. In the literature, some authors use the symbols $\odot$, $\circ$, $\Diamond$, and $\Box$. An index of the notations used in this chapter is given in the appendix.

Manna and Pnueli 1987, Manna and Pnueli 1989]. The combination of $\langle R\rangle$- and $\mathbf{F}^{+}$-operators originates from *dynamic logic* [Salwicki 1970, Pratt 1976] (for an overview on dynamic logics, see [Harel 1984, Kozen and Tiuryn 1990]).

Intuitively, $\mathbf{X}\,\varphi$ indicates that φ holds at some point accessible via a single transition, $\mathbf{F}^{+}\varphi$ specifies that φ must hold in some point which can be reached by a nonempty sequence of transitions, and $\mathbf{F}^{*}\varphi$ means that φ holds at some reachable point (possibly now). Dually, $\cancel{\mathbf{X}}\,\varphi$ holds if all successors satisfy φ, and $\mathbf{G}^{*}\varphi$ and $\mathbf{G}^{+}\varphi$ determine that all reachable points (except maybe the current point) must validate φ. With these operators, example (2) could be written

$$\mathbf{G}^{*}(\mathtt{req} \rightarrow \mathbf{F}^{+}\,\mathtt{ack}).$$

From the definition, $w_0 \models \cancel{\mathbf{X}}\,\varphi$ iff $w_1 \models \varphi$ for all $w_1 \in U$ such that $w_0 \prec w_1$. Similarly, $w_0 \models \mathbf{G}^{+}\varphi$ iff $w_1 \models \varphi$ for all $w_1 \in U$ such that $w_0 < w_1$. A point $w \in U$ is called *terminal*, if $\{w' \mid w \prec w'\} = \{\}$. A terminal point represents a final state of a terminating computation. Terminal points satisfy all $\cancel{\mathbf{X}}$ - and $\mathbf{G}^{+}$-formulas vacuously: if w_0 has no accessible successors, then $w_0 \models \cancel{\mathbf{X}}\,\varphi$ and $w_0 \models \mathbf{G}^{+}\varphi$ for any formula φ.

The difference between $\mathbf{F}^{+}$ and $\mathbf{F}^{*}$ is that in the latter "the future includes the present". Using the $\mathbf{X}$ operator, $\mathbf{F}^{+}$ and $\mathbf{F}^{*}$ can be mutually defined: clearly, the formula $(\mathbf{F}^{*}\varphi \leftrightarrow \varphi \vee \mathbf{F}^{+}\varphi)$ is valid. Therefore, the $\mathbf{F}^{*}$-operator can be expressed by $\mathbf{F}^{+}$. Using the equivalence $(\mathbf{F}^{+}\varphi \leftrightarrow \mathbf{X}\,\mathbf{F}^{*}\varphi)$, each occurrence of the operator $\mathbf{F}^{+}$ in a formula can be replaced by $\mathbf{F}^{*}$ and $\mathbf{X}$, with only a linear increase in formula length. It is not possible to define the $\mathbf{F}^{+}$-operator by $\mathbf{F}^{*}$ alone (without $\mathbf{X}$):

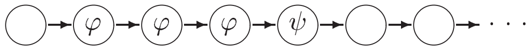
2.1. LEMMA. *Without* $\mathbf{X}$*, the operator* $\mathbf{F}^{+}$ *is strictly more expressive than* $\mathbf{F}^{*}$*.*

PROOF: Consider two models $\mathcal{M}_1$ and $\mathcal{M}_2$, where $U_1 \triangleq U_2 \triangleq \{w\}$, $\mathcal{I}_1(\prec) \triangleq \{\}$, $\mathcal{I}_2(\prec) \triangleq \{(w,w)\}$ and $\mathcal{I}_1(\mathtt{p}) = \mathcal{I}_2(\mathtt{p})$ for all $\mathtt{p} \in \mathcal{P}$. Then $\mathcal{M}_1 \not\models \mathbf{F}^{+}\top$ and $\mathcal{M}_2 \models \mathbf{F}^{+}\top$. However, $w \models \mathbf{F}^{*}\varphi$ iff $w \models \varphi$ in both $\mathcal{M}_1$ and $\mathcal{M}_2$. Therefore, for all formulas φ which involve only propositions, boolean operators and $\mathbf{F}^{*}$ it holds that $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_2 \models \varphi$. (The formal proof of this statement is omitted; it is a straightforward induction on the construction of such formulas.) Hence, there is no formula φ consisting only of propositions, boolean operators and $\mathbf{F}^{*}$ such that for all models $\mathcal{M}$ it holds that $\mathcal{M} \models \varphi$ iff $\mathcal{M} \models \mathbf{F}^{+}\top$. In other words, $\mathbf{F}^{+}\top$ is not expressible in this language. $\square$

A similar proof shows that modal operators cannot express statements about intervals. For example, there is no formula equivalent to example (3) of the above. To remedy this lack of expressiveness, [Kamp 1968] introduced a binary operator $(\varphi\,\mathbf{U}^{+}\psi)$ meaning "φ holds until ψ holds". We use the term *temporal logic* to refer to any modal logic which contains some sort of until-operator. In computer science, this operator was first used by [Gabbay et al. 1980] to classify important properties of concurrent programs. The semantics of $\mathbf{U}^{+}$ is defined as follows:

- $w_0 \models (\varphi\,\mathbf{U}^{+}\psi)$ iff there exists $w_1 \in U$ with $w_0 < w_1$ and $w_1 \models \psi$, and for all $w_2 \in U$ with $w_0 < w_2$ and $w_2 < w_1$, we have $w_2 \models \varphi$.

This situation is illustrated by the following picture.



As an example, the above formula (3) can be expressed with an until-operator as

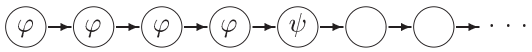
$$\mathbf{G}^{*}(\mathtt{req} \rightarrow (\mathtt{req}\, \mathbf{U}^{+}\, \mathtt{ack})).$$

Various other operators can be defined via $\mathbf{U}^{+}$. Sometime-operator and nexttime operators (for discrete $\prec$) are obtained as follows:

- $\mathbf{X}\, \varphi \leftrightarrow (\bot\, \mathbf{U}^{+}\, \varphi)$
- $\mathbf{F}^{+}\, \varphi \leftrightarrow (\top\, \mathbf{U}^{+}\, \varphi)$

The proof of these equivalences is immediate from the definition: $w_0 \models (\bot\, \mathbf{U}^{+}\, \psi)$ iff there exists $w_1 \in U$ with $w_0 < w_1$ and $w_1 \models \psi$, and for all $w_2 \in U$ with $w_0 < w_2 < w_1$ it holds that $w_2 \models \bot$, which is impossible. In other words, $w_0 < w_1$, but there is no w_2 that satisfies $w_0 < w_2$ and $w_2 < w_1$. Therefore w_1 must be an immediate successor of w_0, i.e., $w_0 \prec w_1$. Consequently, $w_0 \models \mathbf{X}\, \varphi$. The second equivalence is obtained in a similar way.

The *reflexive until*-operator is defined as $(\varphi\, \mathbf{U}^{*}\, \psi) \triangleq (\psi \vee \varphi \wedge (\varphi\, \mathbf{U}^{+}\, \psi))$.



As above, $\mathbf{F}^{*}\, \varphi \leftrightarrow (\top\, \mathbf{U}^{*}\, \varphi)$ and $(\varphi\, \mathbf{U}^{+}\, \psi) \leftrightarrow \mathbf{X}(\varphi\, \mathbf{U}^{*}\, \psi)$. Without $\mathbf{X}$ it is not possible to define $\mathbf{U}^{+}$ or $\mathbf{F}^{+}$ from $\mathbf{U}^{*}$. Hence, $\mathbf{X}$ cannot be defined by $\mathbf{U}^{*}$.

The *unless* or *weak until*-operator is defined as

$$(\varphi\, \mathbf{W}^{+}\, \psi) \triangleq \neg(\neg\psi\, \mathbf{U}^{+}\, \neg(\varphi \vee \psi)).$$

Whereas $(\varphi\, \mathbf{U}^{+}\, \psi)$ requires that ψ eventually holds, $(\varphi\, \mathbf{W}^{+}\, \psi)$ is also true if ψ is never and φ always true. Intuitively, $(\varphi\, \mathbf{W}^{+}\, \psi)$ says that φ holds at least up to the next point where ψ holds. This can be seen as follows: assume that $w_0 \models \neg(\neg\psi\, \mathbf{U}^{+}\, \neg(\varphi \vee \psi))$. By definition, it is not the case that for some $w_1 > w_0$ both $w_1 \models \neg(\varphi \vee \psi)$ and $w_2 \models \neg\psi$ for all $w_0 < w_2 < w_1$. Thus, for all $w_1 > w_0$ it holds that $w_1 \models (\varphi \vee \psi)$, or $w_2 \models \psi$ for some $w_0 < w_2 < w_1$. In other words, if $w_1 > w_0$ then either $w_1 \models \varphi$ or there is some $w_0 < w_2 \leq w_1$ such that $w_2 \models \psi$. Therefore, if $w_2 \not\models \psi$ for all $w_0 < w_2 \leq w_1$, i.e. if w_1 is before the next point where ψ holds, then $w_1 \models \varphi$.

Note that by definition $(\varphi\, \mathbf{W}^{+}\, \bot) = \neg(\top\, \mathbf{U}^{+}\, \neg\varphi) = \mathbf{G}^{+}\, \varphi$. Some texts define the unless operator by $((\varphi\, \mathbf{U}^{+}\, \psi) \vee \mathbf{G}^{+}\, \varphi)$. In *natural models*, which consist of a sequence of points, these two definitions are equivalent:

2.2. Lemma. *For natural models,* $(\varphi\, \mathbf{W}^{+}\, \psi) \leftrightarrow ((\varphi\, \mathbf{U}^{+}\, \psi) \vee \mathbf{G}^{+}\, \varphi)$.

Proof: We must show that for all models $\mathcal{M}$ which are sequences, the following holds: (i) $\mathcal{M} \models ((\varphi\, \mathbf{W}^{+}\, \psi) \rightarrow ((\varphi\, \mathbf{U}^{+}\, \psi) \vee \mathbf{G}^{+}\, \varphi))$, (ii) $\mathcal{M} \models (\mathbf{G}^{+}\, \varphi \rightarrow (\varphi\, \mathbf{W}^{+}\, \psi))$

and (iii) $\mathcal{M} \models ((\varphi\,\mathbf{U}^{+}\,\psi) \to (\varphi\,\mathbf{W}^{+}\,\psi))$. For (i), assume that $w_0 \models (\varphi\,\mathbf{W}^{+}\,\psi)$ and $w_0 \not\models \mathbf{G}^{+}\,\varphi$. Then $w_1 \not\models \varphi$ for some $w_1 > w_0$. According to above, there is some $w_0 < w_2 \leq w_1$ such that $w_2 \models \psi$. Since the model is assumed to be a sequence, it is well-founded. Therefore there must be a smallest w_2 with this property; i.e. $w_0 < w_2 \leq w_1$, $w_2 \models \psi$, and $w_3 \not\models \psi$ for all $w_0 < w_3 < w_2$. Again, according to the above, if $w_0 < w_3 < w_2$ then $w_3 \models \varphi$. Therefore $w_0 \models (\varphi\,\mathbf{U}^{+}\,\psi)$. Formula (ii) follows immediately from the definition: if $w_0 \models \mathbf{G}^{+}\,\varphi$, then $w_1 \models \varphi$ for all $w_1 > w_0$. Therefore, it is not the case that some $w_1 > w_0$ exists which satisfies $w_1 \models \neg(\varphi \vee \psi)$. This implies $w_0 \not\models (\neg\psi\,\mathbf{U}^{+}\,\neg(\varphi \vee \psi))$, i.e., $w_0 \models (\varphi\,\mathbf{W}^{+}\,\psi)$. For implication (iii), we need the property that the model is linear: if $w_0 \models (\varphi\,\mathbf{U}^{+}\,\psi)$, then there exists $w_1 > w_0$ such that $w_1 \models \psi$ and $w_2 \models \varphi$ for all $w_0 < w_2 < w_1$. Assume any point $w > w_0$. Then $w < w_1$ or $w \geq w_1$. In the first case, $w \models \varphi$. In the second case, there exists $w' = w_1$ such that $w' \models \psi$. Thus, for all $w > w_0$ it holds that $w \models \varphi$, or there exists $w_0 < w' \leq w$ such that $w' \models \psi$. This shows that $w_0 \models (\varphi\,\mathbf{W}^{+}\,\psi)$. □

This equivalence does not hold for dense time: for example, if $(U, \prec)$ is isomorphic to the rationals and $\mathcal{I}(\psi) \triangleq \{1/n \mid n \in \mathbf{N}\}$, then $\forall t_1 > 0 \exists t_2 > 0\ (t_2 < t_1 \wedge \psi(t_2))$, hence $0 \models (\bot\,\mathbf{W}^{+}\,\varphi)$. Moreover, $0 \not\models \mathbf{X}\top$ and $0 \models \mathbf{F}^{+}\top$, hence $0 \not\models ((\bot\,\mathbf{U}^{+}\,\psi) \vee \mathbf{G}^{+}\bot)$. For more information on other models of time, see [van Benthem 1991, Gabbay, Hodkinson and Reynolds 1994]. An immediate consequence of Lemma 2.2 is that in natural models the operator $\mathbf{U}^{+}$ is definable by $\mathbf{W}^{+}$ and $\mathbf{F}^{+}$:

$$(\varphi\,\mathbf{U}^{+}\,\psi) \leftrightarrow ((\varphi\,\mathbf{W}^{+}\,\psi) \wedge \mathbf{F}^{+}\,\psi).$$

With first order logic, it is possible to use reverse relations: $x > y$ iff $y < x$. In [Lichtenstein, Pnueli and Zuck 1985], the authors argue that the ability to refer to the past can facilitate program specifications. The temporal *past* or *since*- operator $\mathbf{U}^{-}$ is defined with the following semantics:

- $w_0 \models (\varphi\,\mathbf{U}^{-}\,\psi)$ iff there exists $w_1 \in U$ with $w_1 < w_0$ and $w_1 \models \psi$, and for all $w_2 \in U$ with $w_1 < w_2$ and $w_2 < w_0$, we have $w_2 \models \varphi$.

The syntax of linear temporal logic (**LTL**) is defined as follows:

$$\mathbf{LTL} \quad ::= \quad \mathcal{P} \mid \bot \mid (\mathbf{LTL} \to \mathbf{LTL}) \mid (\mathbf{LTL}\,\mathbf{U}^{+}\,\mathbf{LTL}) \mid (\mathbf{LTL}\,\mathbf{U}^{-}\,\mathbf{LTL}).$$

We write $\mathbf{F}^{-}\,\varphi$ and $\mathbf{G}^{-}\,\varphi$ for $(\top\,\mathbf{U}^{-}\,\varphi)$ and $\neg\,\mathbf{F}^{-}\,\neg\varphi$, respectively. Intuitively, these operators refer to "sometime in the past" and "always in the past". Moreover, $\mathbf{F}^{\pm}\,\varphi$ and $\mathbf{G}^{\pm}\,\varphi$ are abbreviations for $(\mathbf{F}^{-}\,\varphi \vee \varphi \vee \mathbf{F}^{+}\,\varphi)$ and $\neg\,\mathbf{F}^{\pm}\,\neg\varphi$, respectively.

2.3. Expressive Completeness of Temporal Logic

How can first order and temporal logic be compared? Temporal logic can be regarded as a certain fragment of first order logic; this is explained more formally below. In contrast to modal or temporal logics, **FOL** formulas can mention several reference points (free variables). To be able to compare the expressiveness of both type of logics, we restrict **FOL** to formulas with at most one free variable.

The above semantics induces a translation "**FOL**" from modal or temporal to first order logic, where $\mathbf{FOL}(\varphi)$ has exactly one free variable t_0.

- $\mathbf{FOL}(\mathtt{p}) \triangleq \mathtt{p}(t_0)$
- $\mathbf{FOL}(\bot) \triangleq (t_0 \neq t_0)$
- $\mathbf{FOL}((\varphi \to \psi)) \triangleq (\mathbf{FOL}(\varphi) \to \mathbf{FOL}(\psi))$
- $\mathbf{FOL}(\langle R \rangle \varphi) \triangleq \exists t'(t_0 R t' \wedge \mathbf{FOL}(\varphi)\{t_0 := t'\})$
- $\mathbf{FOL}(\mathbf{X}\,\varphi) \triangleq \exists t'(t_0 \prec t' \wedge \mathbf{FOL}(\varphi)\{t_0 := t'\})$
- $\mathbf{FOL}(\mathbf{F}^{+}\,\varphi) \triangleq \exists t'(t_0 < t' \wedge \mathbf{FOL}(\varphi)\{t_0 := t'\})$
- $\mathbf{FOL}(\mathbf{F}^{*}\,\varphi) \triangleq \exists t'(t_0 \leq t' \wedge \mathbf{FOL}(\varphi)\{t_0 := t'\})$
- $\mathbf{FOL}((\varphi\,\mathbf{U}^{+}\,\psi)) \triangleq$
  $\exists t'(t_0 < t' \wedge \mathbf{FOL}(\psi)\{t_0 := t'\} \wedge \forall t''(t_0 < t'' < t' \to \mathbf{FOL}(\varphi)\{t_0 := t''\}))$.
- $\mathbf{FOL}((\varphi\,\mathbf{U}^{-}\,\psi)) \triangleq$
  $\exists t'(t' < t_0 \wedge \mathbf{FOL}(\psi)\{t_0 := t'\} \wedge \forall t''(t' < t'' < t_0 \to \mathbf{FOL}(\varphi)\{t_0 := t''\}))$.

This translation is sometimes called the *standard translation*[Blackburn et al. 2000]. In the translation of $\langle R \rangle \varphi$, ..., $(\varphi\,\mathbf{U}^{+}\,\psi)$, the symbols t' and t'' denote arbitrary variables which do not occur in $\mathbf{FOL}(\varphi)$ or $\mathbf{FOL}(\psi)$. The formula $\mathbf{FOL}(\psi)\{t_0 := t'\}$ denotes the formula $\mathbf{FOL}(\psi)$, where every (free) occurrence of the variable t_0 is replaced by the variable which is denoted by t'. The following example demonstrates the standard translation.

$$\begin{aligned}
&\mathbf{FOL}(((\neg\mathtt{ack}\,\mathbf{U}^{-}\,\mathtt{req})\,\mathbf{U}^{+}\,\mathtt{ack}))\\
&\quad= \exists t_1(t_0 < t_1 \wedge \mathtt{ack}(t_1) \wedge \forall t_2(t_0 < t_2 < t_1 \to \mathbf{FOL}((\neg\mathtt{ack}\,\mathbf{U}^{-}\,\mathtt{req}))\{t_0 := t_2\}))\\
&\quad= \exists t_1(t_0 < t_1 \wedge \mathtt{ack}(t_1) \wedge \forall t_2(t_0 < t_2 < t_1 \to\\
&\qquad\qquad \exists t_3(t_3 < t_2 \wedge \mathtt{req}(t_3) \wedge \forall t_4(t_3 < t_4 < t_2 \to \neg\mathtt{ack}(t_4))))).
\end{aligned}$$

The standard translation of a modal or temporal formula is a first-order formula with exactly one free variable t_0. Correctness of the standard translation can formally be stated as follows:

2.3. Fact. For every $\varphi \in \mathbf{ML}$ or $\mathbf{LTL}$ there exists a first order formula $\mathbf{FOL}(\varphi)$ such that for every frame $(U, \mathcal{I})$, point $w_0 \in U$ and valuation $\mathbf{v}$ for which $\mathbf{v}(t_0) = w_0$ it holds that $(U, \mathcal{I}, w_0) \models \varphi$ iff $(U, \mathcal{I}, \mathbf{v}) \models \mathbf{FOL}(\varphi)$.

Hence, **FOL** is at least as expressive as **LTL**. A logic is called *expressively complete* (or *definitionally complete*), if there exists also a translation in the other direction: given any first-order formula with exactly one free variable, does an equivalent temporal formula exist?

For the translation of any given temporal formula into first order logic only three variables (say, t_0, t_1 and t_2) are really needed. Other variables can be reused; for example, the above $\mathbf{FOL}(((\neg\mathtt{ack}\,\mathbf{U}^{-}\,\mathtt{req})\,\mathbf{U}^{+}\,\mathtt{ack}))$ is equivalent to

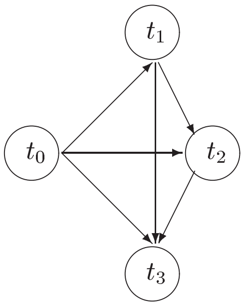
$\exists t_1(t_0 < t_1 \wedge \mathtt{ack}(t_1) \wedge \forall t_2(t_0 < t_2 < t_1 \to$

$$\exists t_0(t_0 < t_2 \wedge \mathtt{req}(t_0) \wedge \forall t_1(t_0 < t_1 < t_2 \rightarrow \neg\mathtt{ack}(t_1))))).$$

Similarly, modal logic can be translated into the so-called *guarded fragment* of first-order logic, which allows only two variables. In the first-order clause for $(\varphi\,\mathbf{U}^+\,\psi)$ three variables are needed. This is the reason why the until-operator is not definable in modal logic. Likewise, **LTL** cannot express any property which "inherently" uses four variables. For example, the statement "there are three different connected points reachable from the current point" is not expressible in temporal logic.

$$\exists t_1, t_2, t_3(t_0 < t_1 \wedge t_0 < t_2 \wedge t_0 < t_3 \wedge t_1 < t_2 \wedge t_1 < t_3 \wedge t_2 < t_3)$$

If $<$ is irreflexive, then a minimal model satisfying this formula is e.g. the following:



In case that $<$ is a linear order (antisymmetric and total) this is equivalent to

$$\exists t_1(t_0 < t_1 \wedge \exists t_2(t_1 < t_2 \wedge \exists t_3(t_2 < t_3)))$$

in which we can rename t_3 by t_0 to get the equivalent

$$\exists t_1(t_0 < t_1 \wedge \exists t_2(t_1 < t_2 \wedge \exists t_0(t_2 < t_0)))$$

which in turn can be expressed temporally as $\mathbf{F}^+\,\mathbf{F}^+\,\mathbf{F}^+\,\top$.

Therefore, attention is restricted to certain classes of structures, like complete linear orders, or finitely-branching trees, etc. A *natural* model consists of a finite or infinite *sequence* of points. Formally, a natural model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$ is a Kripke-model with only one accessibility relation, such that $(U, \prec)$ is isomorphic to the natural numbers or an initial segment of the natural numbers[3], where $\prec$ is the usual successor relation.

2.4. THEOREM (Kamp, Gabbay). *Temporal logic is expressively complete for natural models.*

The original proof of this theorem in [Kamp 1968, pp. 39–94] is extremely complicated. The proof given below follows [Gabbay 1989] and uses a certain property called *separation*. Call a temporal formula

[3]Some textbooks restrict attention to infinite models. Terminating computations are then modelled with an idle loop. In this survey, we use both finite and infinite computation sequences.

- *pure future*, if it is of form $(\varphi\, \mathbf{U}^{+}\, \psi)$, where in both φ and ψ no $\mathbf{U}^{-}$-operator occurs, and
- *pure past*, if it is of form $(\varphi\, \mathbf{U}^{-}\, \psi)$, where in both φ and ψ no $\mathbf{U}^{+}$-operator occurs, and
- *pure present*, if it contains no $\mathbf{U}^{+}$ or $\mathbf{U}^{-}$-operators.

A *future formula* is a boolean combination of pure future and pure present formulas, i.e., one which does not contain any $\mathbf{U}^{-}$-operators. Similarly, a *past formula* does not contain any $\mathbf{U}^{+}$. A formula is *separated* if it is a boolean combination of future and past formulas. A logic has the *separation property* (for a given class of models), if for every formula there exists a separated formula which is equivalent for all models under consideration.

2.5. LEMMA. *The separation property implies expressive completeness.*

PROOF: This lemma is proven by induction on the structure of **FOL**-formulas. For the proof, we assume that **LTL** has the separation property for natural models. That is, for each linear temporal formula there exists an equivalent formula which is separated. We show that any first order formula $\varphi(t_0)$ which has exactly one free variable t_0 can be translated into a temporal formula $\mathbf{LTL}(\varphi)$. It suffices to consider first order logic where $\mathcal{R}^{+} \triangleq \{<, =\}$: in natural models, there is a single accessibility relation, and every atomic subformula $t \prec t'$ can be equivalently replaced by $(t < t' \wedge \neg\exists t''(t < t'' \wedge t'' < t'))$. Furthermore, the scope of quantification can be minimized such that no sub-formula $\varphi \triangleq \exists t\ \psi$ contains a proposition $\mathtt{p}(t')$ where t' is free in φ. For example, $\exists t_1(t_1 > t_0 \wedge \mathtt{p}(t_0) \wedge \mathtt{p}(t_1))$ can be rewritten as $\mathtt{p}(t_0) \wedge \exists t_1(t_1 > t_0 \wedge \mathtt{p}(t_1))$.

The translation of $\mathtt{p}(t_0)$ is $\mathtt{p}$. It is not necessary to give a translation for formulas $\mathtt{p}(t_1)$ or $t_0 \sim t_1$, since they involve other free variables than t_0. The translation of a boolean connective of sub-formulas is the boolean connective of the translation of the sub-formulas. The only remaining case are formulas $\varphi \triangleq \exists t_1\ \psi(t_0, t_1)$. Since the scope of the quantifier $\exists t_1$ is minimal, φ does not contain any proposition $\mathtt{p}(t_0)$. That is, $\psi(t_0, t_1)$ is a boolean combination of formulas $\mathtt{p}(t_1)$, $t_0 \sim t_1$, and $\varphi' \triangleq \exists t_2\ \psi'(t_0, t_1, t_2)$. Replace every sub-formula $t_0 < t$ by a new unary proposition $\mathtt{future}(t)$, replace every sub-formula $t_0 = t$ by a new unary $\mathtt{present}(t)$, and replace every $t < t_0$ by $\mathtt{past}(t)$. That is, φ now does not contain any t_0, and thus each φ' is a formula with exactly one free variable t_1. Since the nesting depth of existential quantifiers in each φ' is smaller than that of φ, we can apply the induction hypothesis to get temporal formulae $\mathbf{LTL}(\varphi')$. Reinserting these into ψ and replacing $\mathtt{p}(t_1)$ in ψ by $\mathtt{p}$, and $\mathtt{q}(t_1)$ by $\mathtt{q}$ for $\mathtt{q} \in \{\mathtt{future}, \mathtt{present}, \mathtt{past}\}$ gives the temporal formula $\mathbf{LTL}(\psi)$. To translate $\varphi \triangleq \exists t_1\ \psi$ we separate the temporal formula $(\mathbf{F}^{-}\, \mathbf{LTL}(\psi) \vee \mathbf{LTL}(\psi) \vee \mathbf{F}^{+}\, \mathbf{LTL}(\psi))$. The resulting formula is a boolean combination of pure future, pure past and pure present formulas. Replace in this formula every $\mathtt{future}$ inside a pure future formula by $\top$, every other $\mathtt{future}$ by $\bot$. Similarly, replace every $\mathtt{past}$ inside a pure past formula by $\top$, and every other $\mathtt{past}$ by $\bot$. Finally, replace every $\mathtt{present}$ inside a pure present formula by $\top$, every other $\mathtt{present}$ by $\bot$. The resulting formula is the required translation $\mathbf{LTL}(\varphi)$.

Given any natural model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$ for φ, define $\mathcal{I}(\mathtt{future}) \triangleq \{w \mid w >$

$w_0\}$, $\mathcal{I}(\texttt{present}) \triangleq \{w_0\}$ and $\mathcal{I}(\texttt{past}) \triangleq \{w \mid w < w_0\}$. Then every step in the above translation preserves validity in $\mathcal{M}$. Therefore, $\mathcal{M} \models \varphi$ iff $\mathcal{M} \models \mathbf{LTL}(\varphi)$. □

To illustrate this construction, let us find the temporal equivalent of $\varphi \triangleq \exists t_1(t_0 < t_1 \wedge \mathtt{p}(t_1) \wedge \forall t_2(t_0 < t_2 < t_1 \rightarrow \mathtt{q}(t_2)))$. (We already know that the outcome should be $(\mathtt{q}\,\mathbf{U}^{+}\,\mathtt{p})$.) The first replacement results in $\exists t_1\psi$, where $\psi \triangleq (\texttt{future}(t_1) \wedge \mathtt{p}(t_1) \wedge \neg\exists t_2(\texttt{future}(t_2) \wedge t_2 < t_1 \wedge \neg\mathtt{q}(t_2)))$. The formula $\varphi'(t_1) = \exists t_2(t_2 < t_1 \wedge \texttt{future}(t_2) \wedge \neg\mathtt{q}(t_2)))$ inductively translates to $\mathbf{LTL}(\varphi') = \mathbf{F}^{-}(\texttt{future} \wedge \neg\mathtt{q}) = \neg\,\mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q})$. Thus $\mathbf{LTL}(\psi) = (\texttt{future} \wedge \mathtt{p} \wedge \mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q}))$. To obtain $\mathbf{LTL}(\exists t_1\psi)$ we have to separate $\mathbf{F}^{\pm}\,\mathbf{LTL}(\psi) = \mathbf{F}^{+}\,\mathbf{LTL}(\psi) \vee \mathbf{LTL}(\psi) \vee \mathbf{F}^{-}\,\mathbf{LTL}(\psi)$. Separating $\mathbf{F}^{+}\,\mathbf{LTL}(\psi) = \mathbf{F}^{+}(\texttt{future} \wedge \mathtt{p} \wedge \mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q}))$ gives $\mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q}) \wedge (\texttt{future} \rightarrow \mathtt{q}) \wedge ((\texttt{future} \rightarrow \mathtt{q})\,\mathbf{U}^{+}(\texttt{future} \wedge \mathtt{p}))$ (see below). The disjuncts $\mathbf{F}^{-}\,\mathbf{LTL}(\psi) = \mathbf{F}^{-}(\texttt{future} \wedge \mathtt{p} \wedge \mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q}))$ and $\mathbf{LTL}(\psi) = (\texttt{future} \wedge \mathtt{p} \wedge \mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q}))$ are already separated. To obtain $\mathbf{LTL}(\varphi)$, we now replace every `future` inside a pure past or pure present formula by $\bot$ and every `future` inside a pure future formula by $\top$. Then $\mathbf{G}^{-}(\texttt{future} \rightarrow \mathtt{q}) \wedge (\texttt{future} \rightarrow \mathtt{q})$ reduces to $\top$, and $((\texttt{future} \rightarrow \mathtt{q})\,\mathbf{U}^{+}(\texttt{future} \wedge \mathtt{p}))$ reduces to $(\mathtt{q}\,\mathbf{U}^{+}\,\mathtt{p})$. The disjuncts $\mathbf{F}^{-}\,\mathbf{LTL}(\psi)$ and $\mathbf{LTL}(\psi)$ reduce to $\bot$. Therefore, $\mathbf{F}^{\pm}\,\mathbf{LTL}(\psi)$ reduces to $(\mathtt{q}\,\mathbf{U}^{+}\,\mathtt{p})$, which is the expected result for $\mathbf{LTL}(\psi)$.

In the above, we used the following equivalence to separate a nested occurrence of future- and past- operators:

$$\models \mathbf{F}^{+}(\varphi \wedge \mathbf{G}^{-}\,\psi) \leftrightarrow \mathbf{G}^{-}\,\psi \wedge \psi \wedge (\psi\,\mathbf{U}^{+}\,\varphi)$$

PROOF: The left side of this formula states that sometimes in the future, φ and always in the past ψ holds. In other words, there is some $w_1 > w_0$ such that φ holds at w_1, and for all $w_2 < w_1$, the formula ψ holds at w_2. In a natural model, each such w_2 must be in the past $(w_2 < w_0)$, present $(w_2 = w_0)$ or future $(w_0 < w_2 < w_1)$ of the current point w_0. Therefore, for each $w_2 < w_0$, the formula ψ holds, and ψ holds at w_0, and there is some $w_1 > w_0$ such that φ holds at w_1, and for all $w_0 < w_2 < w_1$, the formula ψ holds at w_2. This is stated by the right side of the formula. □

A more convenient way to show the correctness of such formulas than by semantical reasoning is by an automated proof procedure. In Section 7, we will show that **LTL** is decidable. There are several automated provers freely available. In fact, the above formula is checked by the STeP system within milliseconds.

To show expressive completeness, it remains to prove the following:

2.6. LEMMA. **LTL** *has the separation property for natural models.*

PROOF: Consider the case of a non-separated formula $\varphi \triangleq (\varphi_1\,\mathbf{U}^{+}\,\varphi_2)$, which contains a direct subformula $\psi \triangleq (\psi_1\,\mathbf{U}^{-}\,\psi_2)$ (i.e., ψ is a boolean component of φ_1 and/or φ_2, and does not occur elsewhere in φ_1 or φ_2). We write $\varphi_i^{\top}$ and $\varphi_i^{\bot}$ for $\varphi_i\{\psi := \top\}$ and $\varphi_i\{\psi := \bot\}$, respectively. By propositional reasoning,

$\varphi_1 \leftrightarrow ((\psi \vee \varphi_1^{\perp}) \wedge (\neg\psi \vee \varphi_1^{\top}))$ and $\varphi_2 \leftrightarrow ((\psi \wedge \varphi_1^{\top}) \wedge (\neg\psi \vee \varphi_2^{\perp}))$. Therefore, φ is equivalent to $(((\psi \vee \varphi_1^{\perp}) \wedge (\neg\psi \vee \varphi_1^{\top}))\, \mathbf{U}^{+} ((\psi \wedge \varphi_2^{\top}) \vee (\neg\psi \wedge \varphi_2^{\perp})))$. By temporal reasoning, this in turn is equivalent to $(((\psi \vee \varphi_1^{\perp})\, \mathbf{U}^{+} (\psi \wedge \varphi_2^{\top})) \vee ((\psi \vee \varphi_1^{\perp})\, \mathbf{U}^{+} (\neg\psi \wedge \varphi_2^{\perp}))) \;\wedge\; (((\neg\psi \vee \varphi_1^{\top})\, \mathbf{U}^{+} (\psi \wedge \varphi_2^{\top})) \;\vee\; ((\neg\psi \vee \varphi_1^{\top})\, \mathbf{U}^{+} (\neg\psi \wedge \varphi_2^{\perp})))$.

For each of the four boolean components of this formula, an equivalent separated formula is given in Fig. 2. Though these formulas are hard to read and difficult to prove manually, their validity can be easily checked by an automated theorem prover. Intuitively, they are generalizations of the example given above. With the separating clauses, φ can be rewritten such that ψ is not in the scope of any $\mathbf{U}^{+}$.

Since the formulas of Fig. 2 still hold if $\mathbf{U}^{+}$ and $\mathbf{U}^{-}$ are interchanged, each $(\varphi_1\, \mathbf{U}^{-}\, \varphi_2)$ containing a direct subformula $\psi \triangleq (\psi_1\, \mathbf{U}^{+}\, \psi_2)$ can be rewritten such that ψ does not occur in the scope of a $\mathbf{U}^{-}$. The general case of several different pasttime-subformulas nested within future-subformulas and vice versa can be handled by repeated application of these transformations. Formally, the claim follows by induction on the nesting depth and number of $\mathbf{U}^{-}$ sub-formulas within $\mathbf{U}^{+}$ and vice versa. □

Since in the separation step of this construction subformulas may be duplicated, the resulting **LTL** formula can be nonelementary larger than the original **FOL** formula.

(i) $(\langle(\psi_1\, \mathbf{U}^{-}\, \psi_2) \vee \varphi_1\rangle\, \mathbf{U}^{+} \langle(\psi_1\, \mathbf{U}^{-}\, \psi_2) \wedge \varphi_2\rangle) \leftrightarrow$
$(\psi_1\, \mathbf{U}^{+}\, \varphi_2) \wedge \langle\psi_2 \vee \psi_1 \wedge (\psi_1\, \mathbf{U}^{-}\, \psi_2)\rangle \vee$
$(\langle\psi_1 \vee \psi_2 \vee \neg(\neg\psi_2\, \mathbf{U}^{+}\, \neg\varphi_1)\rangle\, \mathbf{U}^{+} \langle\psi_2 \wedge (\psi_1\, \mathbf{U}^{+}\, \varphi_2)\rangle) \wedge$
$(\neg(\neg\psi_2\, \mathbf{U}^{+}\, \neg\varphi_1) \vee \langle\psi_2 \vee \psi_1 \wedge (\psi_1\, \mathbf{U}^{-}\, \psi_2)\rangle)$

(ii) $(\langle(\psi_1\, \mathbf{U}^{-}\, \psi_2) \vee \varphi_1\rangle\, \mathbf{U}^{+} \langle\neg(\psi_1\, \mathbf{U}^{-}\, \psi_2) \wedge \varphi_2\rangle) \leftrightarrow$
$(\langle\varphi_1 \wedge \neg\psi_2\rangle\, \mathbf{U}^{+}\, \varphi_2) \wedge \langle(\neg\psi_2 \wedge (\neg\psi_1 \vee \neg(\psi_1\, \mathbf{U}^{-}\, \psi_2)))\rangle \vee$
$(\langle\psi_1 \vee \psi_2 \vee (\varphi_1\, \mathbf{U}^{+} \langle\varphi_2 \vee \varphi_1 \wedge \psi_2\rangle)\rangle\, \mathbf{U}^{+} \langle\neg\psi_1 \wedge \neg\psi_2 \wedge (\langle\varphi_1 \wedge \neg\psi_2\rangle\, \mathbf{U}^{+}\, \varphi_2)\rangle) \wedge$
$(\varphi_1\, \mathbf{U}^{+} \langle\varphi_1 \wedge \psi_2\rangle) \vee \langle\psi_2 \vee \psi_1 \wedge (\psi_1\, \mathbf{U}^{-}\, \psi_2)\rangle$

(iii) $(\langle\neg(\psi_1\, \mathbf{U}^{-}\, \psi_2) \vee \varphi_1\rangle\, \mathbf{U}^{+} \langle(\psi_1\, \mathbf{U}^{-}\, \psi_2) \wedge \varphi_2\rangle) \leftrightarrow$
$(\langle\varphi_1 \wedge \psi_1\rangle\, \mathbf{U}^{+}\, \varphi_2) \wedge \langle\psi_2 \vee \psi_1 \wedge (\psi_1\, \mathbf{U}^{-}\, \psi_2)\rangle \vee$
$(\langle\neg\psi_2 \vee (\varphi_1\, \mathbf{U}^{+} \langle\varphi_2 \vee \varphi_1 \wedge \neg\psi_1 \wedge \neg\psi_2\rangle)\rangle\, \mathbf{U}^{+} \langle\psi_2 \wedge (\langle\varphi_1 \wedge \psi_1\rangle\, \mathbf{U}^{+}\, \varphi_2)\rangle) \wedge$
$([\varphi_1\, \mathbf{U}^{+} \langle\varphi_1 \wedge \neg\psi_1 \wedge \neg\psi_2\rangle] \vee [\neg\psi_2 \wedge \langle\neg\psi_1 \vee \neg(\psi_1\, \mathbf{U}^{-}\, \psi_2)\rangle])$

(iv) $(\langle\neg(\psi_1\, \mathbf{U}^{-}\, \psi_2) \vee \varphi_1\rangle\, \mathbf{U}^{+} \langle\neg(\psi_1\, \mathbf{U}^{-}\, \psi_2) \wedge \varphi_2\rangle) \leftrightarrow$
$(\neg(\psi_1\, \mathbf{U}^{+}\, \neg\varphi_1) \vee \langle\neg\psi_2 \wedge (\neg\psi_1 \vee \neg(\psi_1\, \mathbf{U}^{-}\, \psi_2))\rangle) \wedge$
$(\neg(\langle\psi_1 \vee \psi_2 \vee \neg(\neg\psi_2\, \mathbf{U}^{+}\, \varphi_2)\rangle\, \mathbf{U}^{+} \langle\psi_2 \wedge (\psi_1\, \mathbf{U}^{+}\, \neg\varphi_1)\rangle) \vee$
$((\neg\psi_2\, \mathbf{U}^{+}\, \varphi_2) \wedge \langle\neg\psi_2 \wedge (\neg\psi_1 \vee \neg(\psi_1\, \mathbf{U}^{-}\, \psi_2))\rangle)) \wedge$
$(\mathbf{F}^{+}[\neg\psi_1 \wedge \neg\psi_2 \wedge (\neg\psi_2\, \mathbf{U}^{+}\, \varphi_2)] \vee [(\neg\psi_2\, \mathbf{U}^{+}\, \varphi_2) \wedge (\neg\psi_2 \wedge \langle\neg\psi_1 \vee \neg(\psi_1\, \mathbf{U}^{+}\, \psi_2)\rangle)])$

Figure 2: Separation clauses for **LTL**

3. Second Order Languages

3.1. Linear and Branching Time Logics

As we have seen, linear temporal logic is expressively complete for natural models. The same result (with minor modifications) can be proved for finitely branching trees [Schlingloff 1992*a*, Schlingloff 1992*b*], and for certain partially ordered structures [Thiagarajan and Walukiewicz 1997]. In computer science, the possible executions of a program can be modelled as a *set of execution sequences*. Alternatively, it can be modelled as a unique *execution tree*, where branches denote nondeterministic decisions. This view is adopted in *branching time temporal logic* [Lamport 1980, Ben-Ari, Manna and Pnueli 1983, Emerson and Halpern 1986].

Statements about correctness of program can involve assertions about *all maximal paths* in a tree. A *path* in a model is a (finite or infinite) nonempty sequence of points $\sigma = (w_0, w_1, ...)$, where for each i with $0 \leq i < |\sigma|$ there exists an $R_i \in \mathcal{R}$ such that $(w_i, w_{i+1}) \in \mathcal{I}(R_i)$. A path is *maximal*, if each of its points which has a successor in the model also has a successor in the path. In other words, a maximal path is either infinite, or its final point w_n is terminal (there is no w such that $w_n \prec w$). Computation tree logic (**CTL**) [Clarke and Emerson 1981, Emerson and Clarke 1982] has the following syntax:

$$\mathbf{CTL} \quad ::= \quad \mathcal{P} \;|\; \bot \;|\; (\mathbf{CTL} \rightarrow \mathbf{CTL}) \;|\; \mathbf{E}(\mathbf{CTL}\,\mathbf{U}^{+}\,\mathbf{CTL}) \,|\, \mathbf{A}(\mathbf{CTL}\,\mathbf{U}^{+}\,\mathbf{CTL}).$$

CTL is interpreted on *tree models*. A tree is defined as usual: it has a single root w_0, and every node w_n can be reached from w_0 by exactly one finite path. The transitive closure "$<$" of the successor relation "$\prec$" then denotes the usual tree-order: $(w_1, w_2) \in \mathcal{I}(<)$ iff w_1 is on the (unique) path from the root w_0 up to w_2.

- $w_0 \models \mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi)$ iff there exists $w_1 > w_0$ such that $w_1 \models \psi$, and for all $w_2 \in U$, if $w_0 < w_2 < w_1$ then $w_2 \models \varphi$.
- $w_0 \models \mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)$ iff *for all maximal paths p* from w_0 there exists $w_1 > w_0$ *on path p* such that $w_1 \models \psi$, and for all $w_0 < w_2 < w_1$, $w_2 \models \varphi$.

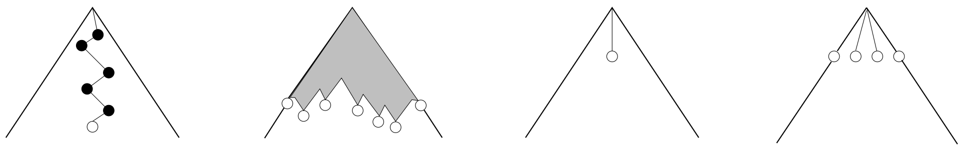
Thus, the $\mathbf{E}\,\mathbf{U}^{+}$-operator is defined similar to the **LTL** until-operator. However, the intended models for **CTL** are trees, whereas **LTL** usually is interpreted on natural models. In **CTL** weak and derived operators can also be defined as abbreviations. However, in branching time, there are two variants of each derived operator.

$$\begin{array}{cc}
\mathbf{E}\,\mathbf{X}\,\psi \triangleq \mathbf{E}(\bot\,\mathbf{U}^{+}\,\psi), & \mathbf{A}\,\mathbf{X}\,\psi \triangleq \mathbf{A}(\bot\,\mathbf{U}^{+}\,\psi), \\
\mathbf{E}\,\cancel{\mathbf{X}}\,\psi \triangleq \neg\mathbf{A}\,\mathbf{X}\,\neg\psi, & \mathbf{A}\,\cancel{\mathbf{X}}\,\psi \triangleq \neg\mathbf{E}\,\mathbf{X}\,\neg\psi, \\[6pt]
\mathbf{E}\,\mathbf{F}^{+}\,\psi \triangleq \mathbf{E}(\top\,\mathbf{U}^{+}\,\psi), & \mathbf{A}\,\mathbf{F}^{+}\,\psi \triangleq \mathbf{A}(\top\,\mathbf{U}^{+}\,\psi), \\
\mathbf{E}\,\mathbf{G}^{+}\,\psi \triangleq \neg\mathbf{A}\,\mathbf{F}^{+}\,\neg\psi, & \mathbf{A}\,\mathbf{G}^{+}\,\psi \triangleq \neg\mathbf{E}\,\mathbf{F}^{+}\,\neg\psi, \\[6pt]
\mathbf{E}(\varphi\,\mathbf{U}^{*}\,\psi) \triangleq (\psi \vee \varphi \wedge \mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi)), & \mathbf{A}(\varphi\,\mathbf{U}^{*}\,\psi) \triangleq (\psi \vee \varphi \wedge \mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)), \\
\mathbf{E}\,\mathbf{F}^{*}\,\psi \triangleq (\psi \vee \mathbf{E}\,\mathbf{F}^{+}\,\psi), & \mathbf{A}\,\mathbf{F}^{*}\,\psi \triangleq (\psi \vee \mathbf{A}\,\mathbf{F}^{+}\,\psi), \\
\mathbf{E}\,\mathbf{G}^{*}\,\psi \triangleq (\psi \wedge \mathbf{E}\,\mathbf{G}^{+}\,\psi), & \mathbf{A}\,\mathbf{G}^{*}\,\psi \triangleq (\psi \wedge \mathbf{A}\,\mathbf{G}^{+}\,\psi),
\end{array}$$

$$\mathbf{E}(\varphi\,\mathbf{W}^{+}\,\psi) \triangleq \neg\mathbf{A}(\neg\psi\,\mathbf{U}^{+}\,\neg(\varphi\vee\psi)), \qquad \mathbf{A}(\varphi\,\mathbf{W}^{+}\,\psi) \triangleq \neg\mathbf{E}(\neg\psi\,\mathbf{U}^{+}\,\neg(\varphi\vee\psi)).$$

Informally, $\mathbf{E}\,\mathbf{X}\,\psi$ means that some successor node satisfies ψ, and $\mathbf{A}\,\cancel{\mathbf{X}}\,\psi$ holds if all successors are ψ. In a terminal point, $\mathbf{A}\,\cancel{\mathbf{X}}\perp$ is valid, but $\mathbf{A}\,\mathbf{X}\perp$ not: if w_0 has no successors, then the only maximal path p from w_0 is the one-element sequence $\sigma = (w_0)$. On this unique path σ there is no $w_1 > w_0$, therefore each formula $\mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)$ and $\mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi)$ must be invalid. As a special case, in such a point $\mathbf{E}\,\mathbf{X}\top$ is not valid, but $\mathbf{E}\,\cancel{\mathbf{X}}\top$ and $\mathbf{E}\,\cancel{\mathbf{X}}\perp$ are valid. In a nonterminal point, $(\mathbf{E}\,\mathbf{X}\,\varphi \leftrightarrow \mathbf{E}\,\cancel{\mathbf{X}}\,\varphi)$ and $(\mathbf{A}\,\mathbf{X}\,\varphi \leftrightarrow \mathbf{A}\,\cancel{\mathbf{X}}\,\varphi)$. Thus, if we restrict attention to models without terminal points, these operators coincide. The operators $\mathbf{A}\,\mathbf{X}$ and $\mathbf{E}\,\cancel{\mathbf{X}}$ can be expressed by $\mathbf{E}\,\mathbf{X}$ and $\mathbf{A}\,\cancel{\mathbf{X}}$ (with at most linear increase of formula length) via $(\mathbf{A}\,\mathbf{X}\,\varphi \leftrightarrow \mathbf{A}\,\cancel{\mathbf{X}}\,\varphi \wedge \mathbf{E}\,\mathbf{X}\top)$ and $(\mathbf{E}\,\cancel{\mathbf{X}}\,\varphi \leftrightarrow \mathbf{E}\,\mathbf{X}\,\varphi \vee \mathbf{A}\,\cancel{\mathbf{X}}\perp)$, that is, $(\mathbf{E}\,\cancel{\mathbf{X}}\,\varphi \leftrightarrow (\mathbf{E}\,\mathbf{X}\top \rightarrow \mathbf{E}\,\mathbf{X}\,\varphi))$. Thus, all **CTL** nexttime-operators can be expressed in terms of $\mathbf{E}\,\mathbf{X}$.

The formula $\mathbf{E}\,\mathbf{F}^{*}\,\psi$ means that some node in the computation tree satisfies ψ, and $\mathbf{A}\,\mathbf{F}^{*}\,\psi$ specifies that ψ must hold somewhere along every maximal computation path. Dually, $\mathbf{A}\,\mathbf{G}^{*}\,\psi$ means that every node in the (sub-) tree satisfies ψ, whereas $\mathbf{E}\,\mathbf{G}^{*}\,\psi$ indicates that ψ is globally valid along some path.



$\mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi)$ $\quad$ $\mathbf{A}(\varphi\,\mathbf{U}^{*}\,\psi)$ $\quad$ $\mathbf{E}\,\mathbf{X}\,\psi$ $\quad$ $\mathbf{A}\,\mathbf{X}\,\psi$

In the above picture, nodes satisfying φ are shown solid (or as a shaded area), whereas ψ nodes are indicated by a circle.

The operator $\mathbf{A}\,\mathbf{U}^{+}$ can be expressed by $\mathbf{E}\,\mathbf{U}^{+}$ and $\mathbf{A}\,\mathbf{F}^{+}$. This characterization is similar to the definition of the unless-operator in linear temporal logic, cf. page 1648:

$$\mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi) \leftrightarrow (\mathbf{A}(\varphi\,\mathbf{W}^{+}\,\psi) \wedge \mathbf{A}\,\mathbf{F}^{+}\,\psi) = (\neg\mathbf{E}(\neg\psi\,\mathbf{U}^{+}\,\neg(\varphi\vee\psi)) \wedge \mathbf{A}\,\mathbf{F}^{+}\,\psi).$$

Therefore, it is sufficient to consider only the two basic operators $\mathbf{E}\,\mathbf{U}^{+}$ and $\mathbf{A}\,\mathbf{F}^{+}$ in formal proofs and algorithms. Similarly, the formula $\mathbf{E}(\varphi\,\mathbf{W}^{+}\,\psi)$ can be replaced by $(\mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi) \vee \mathbf{E}\,\mathbf{G}^{+}\,\varphi)$. However, there is no negation-free "dual" characterization of $\mathbf{A}\,\mathbf{W}^{+}$ and $\mathbf{E}\,\mathbf{U}^{+}$.

We now give some examples of **CTL** formulas. The following properties are typical correctness requirements that might arise in the verification of a finite state concurrent program.

— $\mathbf{E}\,\mathbf{F}^{+}(\texttt{started} \wedge \neg\texttt{ready})$: it is possible to get to a state where `started` holds but `ready` does not hold.

— $\mathbf{A}\,\mathbf{G}^{*}(\texttt{req} \rightarrow \mathbf{A}\,\mathbf{F}^{+}\,\texttt{ack})$: if a request occurs, then it will be eventually acknowledged

— $\mathbf{A}\,\mathbf{G}^{*}\ \mathbf{A}\,\mathbf{F}^{*}$ `stack_is_empty`: the proposition `stack_is_empty` holds infinitely often on every computation path

— $\mathbf{A}\,\mathbf{G}^{*}\ \mathbf{E}\,\mathbf{F}^{*}$ `restart`: from any state it is possible to get to a `restart` state.

For many **CTL** formulas it is possible to formulate similar correctness properties in **LTL**. *Possibility properties* like the last one mentioned above can not be formulated in **LTL**. On the other hand, certain *fairness properties* cannot be formulated in **CTL**.

How can we compare the expressivity of **CTL** with (the future fragment of) **LTL**? Direct comparison is difficult, since models are different: on natural models, which are special tree models with branching degree one, $\mathbf{A}\,\mathbf{U}^{+}$ and $\mathbf{E}\,\mathbf{U}^{+}$-operators coincide. On tree models with higher branching degree, **LTL** obviously cannot express $\mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)$.

Therefore, one considers **LTL** and **CTL** on (nonlinear, non-tree) Kripke-models $(U, \mathcal{I}, w_0)$. In contrast to natural or tree models, Kripke-models can contain reflexive points, loops or even dense relations. We call an **LTL** future formula *sequence-valid* in a Kripke-model $\mathcal{M}$, if it is valid in all natural models $((w_0, w_1, ...), \mathcal{I}, w_0)$ which are *generated* from $\mathcal{M}$, that is, for all maximal paths $w_0, w_1, ...)$ in U starting from w_0. (A formal definition of this notion will be given in Section 4.) Similarly, a **CTL**-formula is called *tree-valid* in a Kripke-model, if it is valid in the root of the unique maximal tree generated from it.

With this definition, the expressivity of **LTL** and **CTL** can be compared. It turns out that on Kripke models, neither of both is strictly more expressive than the other one. For example, the **LTL** formula $\varphi \triangleq \mathbf{F}^{+}\,\mathbf{G}^{+}\,\mathtt{p}$ is not expressible in **CTL** (it is *not* the same property as $\mathbf{A}\,\mathbf{F}^{+}\,\mathbf{A}\,\mathbf{G}^{+}\,\mathtt{p}$). That is, there is no **CTL**-formula ψ such that ψ is tree-valid in exactly the same Kripke-models in which φ is sequence-valid. Similarly, $\mathbf{A}\,\mathbf{G}^{+}\,\mathbf{E}\,\mathbf{F}^{+}\,\mathtt{p}$ is not expressible in **LTL** (it is *not* the same as $\mathbf{G}^{+}\,\mathbf{F}^{+}\,\mathtt{p}$). For more information on the expressiveness of linear versus branching time see [Emerson and Lei 1985, Emerson and Halpern 1986, Clarke and Draghicescu 1988, Emerson 1990].

On Kripke-models, the logic $\mathbf{CTL}^*$ (see [Emerson and Lei 1985, Emerson and Halpern 1986]) subsumes **CTL** and **LTL** by separating path quantification ($\mathbf{E}$) from temporal quantification ($\mathbf{U}^{+}$). Thus it is possible to write e.g. $\mathbf{E}\,\mathbf{G}^{*}\,\mathbf{F}^{*}\,\mathtt{p}$. The logic $\mathbf{CTL}^*$ is strictly more expressive than both **CTL** and **LTL**. On binary trees, the expressiveness of $\mathbf{CTL}^*$ can be compared to first order logic with additional (second order) quantification on paths. For more information on the expressiveness and complexity of various sublogics of $\mathbf{CTL}^*$, see [Emerson 1990].

3.2. Propositionally Quantified Logics

Quantification over maximal paths is not a first-order notion. It is clear that for natural models, which consist of exactly one maximal path, this quantifier is not very useful. However, even for natural models, there might be other types of second-order quantification which could be interesting. Wolper remarked that "temporal logic can be more expressive"[Wolper 1982, Wolper 1983]. In temporal or first-order logic, it is not possible to specify that a certain proposition $\mathtt{p}$ holds on every *second* point of an execution sequence, without constraining the values of $\mathtt{p}$ in intermediate points. Formally, for a natural model where $U = (w_0, w_1, ...)$, define the new operator $\mathbf{G}^{2n}$ by

$$w_i \models \mathbf{G}^{2n}\,\varphi \quad \text{iff} \quad w_{i+2n} \models \varphi \text{ for all } n \geq 0$$

We will show that this operator can not be expressed in **LTL** or **FOL**. First, note that the following operators are not equivalent to $\mathbf{G}^{2n}\,\varphi$.

$$\mathbf{G}^{2n}_{\mathbf{LTL}}\,\varphi \triangleq \varphi \wedge \mathbf{G}^{*}(\varphi \rightarrow \bar{\mathbf{X}}\,\bar{\mathbf{X}}\,\varphi)$$
$$(\mathbf{G}^{2n}_{\mathbf{FOL}}\,\varphi)(t_0) \triangleq \varphi(t_0) \wedge \forall t \geq t_0(\varphi(t) \rightarrow \forall t_1, t_2(t \prec t_1 \prec t_2 \rightarrow \varphi(t_2)))$$

These formulas define a stronger property than required: they imply that if φ holds in two adjacent states, it must hold always. Therefore, $\models (\mathbf{G}^{2n}_{\mathbf{LTL}}\,\varphi \rightarrow \mathbf{G}^{2n}\,\varphi)$. The reverse implication does not hold: there are models satisfying $\mathbf{G}^{2n}\,\varphi$ but not $\mathbf{G}^{2n}_{\mathbf{LTL}}\,\varphi$ or $\mathbf{G}^{2n}_{\mathbf{FOL}}\,\varphi(t_0)$, respectively.

3.1. THEOREM (Wolper). *Let* $\mathtt{p}$ *be any atomic proposition. There is no* **LTL**-*formula* φ *such that* $\models \varphi \leftrightarrow \mathbf{G}^{2n}\,\mathtt{p}$.

PROOF: Consider the following sequence $(\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, ...)$ of models. For each $i \geq 0$, define $\mathcal{M}_i \triangleq (U_i, \mathcal{I}_i, w^i_0)$, where $(U_i, \prec)$ is isomorphic to the integers: $U_i \triangleq (..., w^i_{-2}, w^i_{-1}, w^i_0, w^i_1, w^i_2, ...)$. Furthermore, define $\mathcal{I}_i(\mathtt{q}) \triangleq U_i \setminus w^i_i$ for all $\mathtt{q} \in \mathcal{P}$. That is, $w^i_n \models \mathtt{q}$ iff $i \neq n$ for all atomic propositions $\mathtt{q}$. Since $(U_i, \mathcal{I}_i, w^i_0)$ is isomorphic to $(U_{i+1}, \mathcal{I}_{i+1}, w^{i+1}_1)$, we have $w^i_0 \models \varphi$ iff $w^{i+1}_1 \models \varphi$ for all formulas φ. As a consequence, $w^i_0 \models \varphi$ iff $w^{i+1}_0 \models \mathbf{X}\,\varphi$.

In the next step, we prove that any **LTL** formula will almost always be **true** or almost always be **false** in the sequence $(\mathcal{M}_i)$: for any $\varphi \in \mathbf{LTL}$ there exists an i such that for all $j \geq i$ it holds that $\mathcal{M}_i \models \varphi$ iff $\mathcal{M}_j \models \varphi$. This is proved by induction on the structure of **LTL** formulas. The only interesting case is given by the until-connectives. We prove the case of $(\varphi\,\mathbf{U}^{*}\,\psi)$. For this case, the induction hypothesis guarantees that there is an i such that for all $j \geq i$, both $w^j_0 \models \varphi$ iff $w^{j+1}_0 \models \varphi$ (*) and $w^j_0 \models \psi$ iff $w^{j+1}_0 \models \psi$ (**). We have to show that $w^j_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$ iff $w^{j+1}_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$. From the above consequence, $w^j_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$ iff $w^{j+1}_0 \models \mathbf{X}(\varphi\,\mathbf{U}^{*}\,\psi)$ (***). The following recursive characterization is valid: $\models (\varphi\,\mathbf{U}^{*}\,\psi) \leftrightarrow (\psi \vee \varphi \wedge \mathbf{X}(\varphi\,\mathbf{U}^{*}\,\psi))$. In particular, this implies $\models (\psi \rightarrow (\varphi\,\mathbf{U}^{*}\,\psi))$ (†), $\models (\neg\psi \rightarrow ((\varphi\,\mathbf{U}^{*}\,\psi) \leftrightarrow (\varphi \wedge \mathbf{X}(\varphi\,\mathbf{U}^{*}\,\psi))))$ (††), and $\models (\neg\psi \rightarrow ((\varphi\,\mathbf{U}^{*}\,\psi) \rightarrow \varphi))$ († † †).

If $w^j_0 \models \psi$, then $w^j_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$ by (†). In this case, by (**), $w^{j+1}_0 \models \psi$, hence also $w^{j+1}_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$ by (†). Therefore, if $w^j_0 \models \psi$, then $w^j_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$ iff $w^{j+1}_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$. Now we consider the case that $w^j_0 \not\models \psi$. By († † †), $w^j_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$ iff $w^j_0 \models \varphi$ *and* $w^j_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$. By (*) and (***), this in turn holds iff $w^{j+1}_0 \models \varphi$ and $w^{j+1}_0 \models \mathbf{X}(\varphi\,\mathbf{U}^{*}\,\psi)$. By (††), this is the case iff $w^{j+1}_0 \models (\varphi\,\mathbf{U}^{*}\,\psi)$.

To complete the proof, we now show that this eventual stability property does not hold for formulas which include the $\mathbf{G}^{2n}$ operator. It is not hard to see that $\mathcal{M}_i \models \mathbf{G}^{2n}\,\mathtt{p}$ iff i is odd: recall that $w^i_i \not\models \mathtt{p}$. Thus, if i is even, then for $n \triangleq i/2$

we have $w^i_{0+2n} \not\models \mathtt{p}$, which means $w^i_0 \not\models \mathbf{G}^{2n}\,\mathtt{p}$. If i is odd, however, then for all $n \geq 0$, $w^i_{0+2n} \models \mathtt{p}$, and thus $w^i_0 \models \mathbf{G}^{2n}\,\mathtt{p}$. Hence, we have shown that for every **LTL** formula φ there is a model $\mathcal{M}_i$ such that $\mathcal{M}_i \not\models (\varphi \leftrightarrow \mathbf{G}^{2n}\,\mathtt{p})$. ☐

The above proof shows that the $\mathbf{G}^{2n}$ operator cannot be defined in the basic temporal or first order language. However, it can be defined if additional propositions are allowed. To assert that $\mathbf{G}^{2n}\,\varphi$ holds, it suffices to provide a "new" proposition $\mathtt{q}$ (not occurring in φ) such that $\mathbf{G}^{2n}_{\mathbf{LTL}}\,\mathtt{q}$ holds, and that φ is valid wherever $\mathtt{q}$ is valid. This puts an additional constraint on the "auxiliary variable" $\mathtt{q}$, which can be considered as an "implementation detail" in the context of φ. If we disregard the value of $\mathtt{q}$, then the models satisfying $(\mathbf{G}^{2n}_{\mathbf{LTL}}\,\mathtt{q} \wedge \mathbf{G}^{*}(\mathtt{q} \rightarrow \varphi))$ are exactly those satisfying $\mathbf{G}^{2n}\,\varphi$. That is, for any model $\mathcal{M}$ such that $\mathcal{M} \models (\mathbf{G}^{2n}_{\mathbf{LTL}}\,\mathtt{q} \wedge \mathbf{G}^{*}(\mathtt{q} \rightarrow \varphi))$ it holds that $\mathcal{M} \models \mathbf{G}^{2n}\,\varphi$, and for every model $\mathcal{M}$ such that $\mathcal{M} \models \mathbf{G}^{2n}\,\varphi$ it holds that $\mathcal{M}' \models (\mathbf{G}^{2n}_{\mathbf{LTL}}\,\mathtt{q} \wedge \mathbf{G}^{*}(\mathtt{q} \rightarrow \varphi))$, where $\mathcal{M}'$ differs from $\mathcal{M}$ only in the fact that $\mathcal{I}(\mathtt{q}) = \{w_0, w_2, w_4, ...\}$. Logically, this projection operation amounts to existential quantification on temporal propositions or sets of points:

$$\mathbf{G}^{2n}\,\varphi \leftrightarrow \exists q(\mathbf{G}^{2n}_{\mathbf{LTL}}\,q \wedge \mathbf{G}^{*}(q \rightarrow \varphi))$$
$$(\mathbf{G}^{2n}\,\varphi)(t_0) \leftrightarrow \exists q((\mathbf{G}^{2n}_{\mathbf{FOL}}\,q)(t_0) \wedge \forall t \geq t_0(q(t) \rightarrow \varphi(t))))$$

The language used in the first of these formulas is called quantified temporal logic **qTL** [Sistla 1983], the language of the second item is *monadic second order logic* **MSOL**.

$$\begin{aligned}\mathbf{qTL} ::= {} & \mathcal{P} \mid \mathcal{Q} \mid \bot \mid (\mathbf{qTL} \rightarrow \mathbf{qTL}) \mid \\ & (\mathbf{qTL}\,\mathbf{U}^{+}\,\mathbf{qTL}) \mid (\mathbf{qTL}\,\mathbf{U}^{-}\,\mathbf{qTL}) \mid \exists\mathcal{Q}\;\mathbf{qTL}.\end{aligned}$$

$$\begin{aligned}\mathbf{MSOL} ::= {} & \mathcal{P}(\mathcal{T}) \mid \mathcal{Q}(\mathcal{T}) \mid \bot \mid (\mathbf{MSOL} \rightarrow \mathbf{MSOL})\mid \\ & \mathcal{R}^{+}(\mathcal{T},\mathcal{T}) \mid \exists\mathcal{T}\;\mathbf{MSOL} \mid \exists\mathcal{Q}\;\mathbf{MSOL}\end{aligned}$$

To define this syntax, we used another syntactic category $\mathcal{Q} = \{q, q_0, ...\}$ of *proposition variables*. Any valuation in a model $\mathbf{v}$ assigns a set $\mathbf{v}(q) \subseteq U$ to each of these (second order) variables. The formula $\exists q\;\varphi$ is valid in a model $\mathcal{M} = (U, \mathcal{I}, \mathbf{v})$ if it is valid in some model $\mathcal{M}' = (U, \mathcal{I}, \mathbf{v}')$ which differs from $\mathcal{M}$ at most in the valuation of the proposition variable $q \in \mathcal{Q}$.

It is easy to lift the expressive completeness theorem 2.4 to second order.

3.2. LEMMA. *On natural models,* **qTL** *has the same expressiveness as* **MSOL**.

PROOF: In the proof of Theorem 2.4, it was shown how to construct the translation $\mathbf{LTL}(\varphi)$ of a first order formula φ. For any **MSOL** formula there is an equivalent prenex formula of the form $\sigma q_1 \sigma q_2 ... \sigma q_n \psi$, where ψ is a first order formula and each σ is a second order quantifier. Thus, defining $\mathbf{MSOL}(\sigma q_1 \sigma q_2 ... \sigma q_n \psi)$ by $\sigma q_1 \sigma q_2 ... \sigma q_n\ \mathbf{LTL}(\psi)$ gives a translation from **MSOL** into **qTL**. ☐

3.3. Lemma. *On natural models, the* $\mathbf{U}^{+}$*-operator in* **qTL** *is definable by the operators* $\mathbf{G}^{\pm}$ *and* $\mathbf{X}$*:*

$$(\varphi\,\mathbf{U}^{+}\,\psi) \leftrightarrow \forall q(\mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q) \to q).$$

Proof: Since this lemma is used several times in subsequent sections, we give a detailed proof. For one direction, assume that $(\varphi\,\mathbf{U}^{+}\,\psi)$ is valid in $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$. To prove that $\mathcal{M} \models \forall q(\mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q) \to q)$, let $\mathcal{I}'(q)$ be an arbitrary set of points, and show that $(U, \mathcal{I}', w_0) \models (\mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q) \to q)$. In other words, from the assumption $w_0 \models \mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q)$ we have to show that $w_0 \models q$. In any natural model satisfying $w_0 \models (\varphi\,\mathbf{U}^{+}\,\psi)$, there are w_1, ..., $w_n \in U$ such that $w_i \prec w_{i+1}$ for all $0 \leq i < n$, and $\varphi(w_i)$ for all $0 < i < n$, and $w_n \models \psi$. If $w_0 \models \mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q)$, then $w_i \models (\mathbf{X}(\psi \vee \varphi \wedge q) \to q)$ for all $i \geq 0$. Hence, $w_i \models (\mathbf{X}\,\psi \to q)$ and $w_i \models (\mathbf{X}(\varphi \wedge q) \to q)$ for all $i \geq 0$. From $w_n \models \psi$ it follows that $w_{n-1} \models \mathbf{X}\,\psi$. Since $w_{n-1} \models (\mathbf{X}\,\psi \to q)$, we have $w_{n-1} \models q$. Therefore $w_{n-1} \models (\varphi \wedge q)$, and $w_{n-2} \models \mathbf{X}(\varphi \wedge q)$. Since $w_{n-2} \models (\mathbf{X}(\varphi \wedge q) \to q)$, it follows that $w_{n-2} \models q$. Continuing inductively, we find that $w_i \models q$ for all $0 \leq i < n$. Therefore, $w_0 \models q$.

For the other direction, assume that $w_0 \models \forall q(\mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q) \to q)$ and show that $w_0 \models (\varphi\,\mathbf{U}^{+}\,\psi)$. First, we show that there must be some $w > w_0$ satisfying $w \models \psi$. Assume for contradiction that this is not the case. Choose $\mathcal{I}(q) \triangleq \{w \mid \text{not } w \geq w_0\}$. In natural models, this is the set $\{w \mid w < w_0\}$. It follows that (i) $w \models q$ for all w such that not $w \geq w_0$, (ii) $w_0 \not\models q$, and (iii) $w \not\models q$ for all $w > w_0$. We show that (*): $w \models (\mathbf{X}(\psi \vee \varphi \wedge q) \to q)$ for all $w \in U$. According to the contradiction assumption, $w \not\models \psi$ for all $w > w_0$. With (iii), it follows that $w \not\models (\psi \vee \varphi \wedge q)$ for all $w > w_0$. Hence, $w \not\models \mathbf{X}(\psi \vee \varphi \wedge q)$ for all $w \geq w_0$. As a consequence, (*) holds for all $w \geq w_0$. If not $w \geq w_0$, then (*) is an immediate consequence of (i). From (*), we infer that $w_0 \models \mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q)$. Therefore, $w_0 \models q$, which is a contradiction to (ii).

Let $w_1, ..., w_n$ be a set of points such that $w_i \prec w_{i+1}$ for all $0 \leq i < n$, and w_n is the smallest point satisfying ψ (i.e., $w_n \models \psi$ and $w_i \models \neg\psi$ for all $w_0 < w_i < w_n$). If $n = 1$, we are done: in this case $w_0 \models \mathbf{X}\,\psi$, which implies that $w_0 \models (\varphi\,\mathbf{U}^{+}\,\psi)$. If $n > 1$, to prove $w_0 \models (\varphi\,\mathbf{U}^{+}\,\psi)$ we additionally have to show that $w_i \models \varphi$ for any $0 < i < n$. Substitution of q with $\neg q$ in the assumption yields the following equivalent version: $w_0 \models \forall q(q \to \mathbf{F}^{\pm}(q \wedge \mathbf{X}(\psi \vee \varphi \wedge \neg q)))$. Choose $\mathcal{I}(q) \triangleq \{w \mid w_0 \leq w < w_i\}$. It follows that $w_0 \models \mathbf{F}^{\pm}(q \wedge \mathbf{X}(\psi \vee \varphi \wedge \neg q))$. That is, there is some $w \in U$ such that $w \models (q \wedge \mathbf{X}(\psi \vee \varphi \wedge \neg q))$. Since n is minimal, there is no $w \in \mathcal{I}(q)$ which satisfies $w \models \mathbf{X}\,\psi$. Therefore, it follows that there is a $w \geq w_0$ such that $w \models (q \wedge \mathbf{X}(\varphi \wedge \neg q))$. Since w_{i-1} is the only point with $w_{i-1} \models (q \wedge \mathbf{X}\,\neg q)$ we can conclude that $w_{i-1} \models \mathbf{X}\,\varphi$, i.e., $w_i \models \varphi$. □

As a sideline we remark that this proof does not make essential use of the "past-component" of the $\mathbf{G}^{\pm}$-operator; in fact, the same proof holds verbatim if we replace $\mathbf{G}^{\pm}$ by $\mathbf{G}^{*}$ and $\mathbf{F}^{\pm}$ by $\mathbf{F}^{*}$. Thus, a corollary to Lemma 3.3 is $(\varphi\,\mathbf{U}^{+}\,\psi) \leftrightarrow \forall q(\mathbf{G}^{*}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q) \to q)$. (Since $\mathbf{F}^{+}$ is somewhat more specific than $\mathbf{F}^{\pm}$ this could be considered as a somehow weaker result.)

The characterization of the $\mathbf{U}^{+}$-operator with second order quantification is a special case of the general scheme $\forall q(\mathbf{G}^{\pm}(\xi \rightarrow q) \rightarrow q)$, where $\xi \triangleq \mathbf{X}(\psi \vee \varphi \wedge q)$. Dually, the operator $(\varphi\, \mathbf{W}^{+}\, \psi) \triangleq \neg(\neg\psi\, \mathbf{U}^{+}\, \neg(\varphi \vee \psi))$ is characterized by

$$
\begin{aligned}
(\varphi\, \mathbf{W}^{+}\, \psi) &\leftrightarrow \neg\forall q(\mathbf{G}^{\pm}(\mathbf{X}(\neg(\varphi \vee \psi) \vee (\neg\psi \wedge q)) \rightarrow q) \rightarrow q)\\
&\leftrightarrow \exists q(\neg q \wedge \mathbf{G}^{\pm}(\mathbf{X}((\neg\psi \wedge \neg\varphi) \vee (\neg\psi \wedge q)) \rightarrow q))\\
&\leftrightarrow \exists q(\neg q \wedge \mathbf{G}^{\pm}(\neg q \rightarrow \neg\, \mathbf{X}(\neg\psi \wedge (\neg\varphi \vee q))))\\
&\leftrightarrow \exists q(\neg q \wedge \mathbf{G}^{\pm}(\neg q \rightarrow \cancel{\mathbf{X}}(\psi \vee (\varphi \wedge \neg q))))\\
&\leftrightarrow \exists q(q \wedge \mathbf{G}^{\pm}(q \rightarrow \cancel{\mathbf{X}}(\psi \vee \varphi \wedge q)))
\end{aligned}
$$

This is an instance of the dual scheme $\exists q(q \wedge \mathbf{G}^{\pm}(q \rightarrow \xi))$ with $\xi \triangleq \cancel{\mathbf{X}}(\psi \vee \varphi \wedge q)$.

For complexity reasons, it is not always advisable to allow quantifiers on arbitrary subsets of the universe U. Therefore, we introduce *fixpoint quantification*: quantification on sets which follows these schemes. This results in the *propositional μ-calculus* $\mu\mathbf{TL}$ [Emerson and Clarke 1980, Pratt 1981, Kozen 1983, Kozen and Parikh 1983]:

$$\mu\mathbf{TL} \quad ::= \quad \mathcal{P} \quad | \quad \mathcal{Q} \quad | \quad \bot \quad | \quad (\mu\mathbf{TL} \rightarrow \mu\mathbf{TL}) \quad | \quad \langle\mathcal{R}\rangle\ \mu\mathbf{TL} \quad | \quad \nu\mathcal{Q}\ \mu\mathbf{TL}.$$

The semantics of $\mu\mathbf{TL}$ can be defined by a translation into **MSOL**.

- $\mathbf{MSOL}(\varphi)$ is defined as in $\mathbf{FOL}(\varphi)$, for the cases $\mathtt{p} \in \mathcal{P}$, $\bot$, $(\psi_1 \rightarrow \psi_2)$, and $\langle\mathcal{R}\rangle\,\psi$
- $\mathbf{MSOL}(q) \triangleq q(t_0)$, if $q \in \mathcal{Q}$
- $\mathbf{MSOL}(\nu q\ \varphi) \triangleq \exists q(q(t_0) \wedge \forall t(q(t) \rightarrow \mathbf{MSOL}(\varphi)\{t_0 := t\}))$.

Recall that $\varphi\{t_0 := t\}$ denotes the formula which is formed from φ by replacing every free occurrence of t_0 by t. Similarly, $\varphi\{q := \psi\}$ denotes the formula which results from φ by replacing every free occurrence of q with ψ. The formula $\mu q\ \varphi$ is short for $\neg\nu q\ \neg(\varphi\{q := \neg q\})$. Thus, the translation of $\mu q\ \varphi$ evaluates to

- $$\begin{aligned}\mathbf{MSOL}(\mu q\ \varphi) &= \neg\exists q(\neg q(t_0) \wedge \forall t(\neg q(t) \rightarrow \neg\mathbf{MSOL}(\varphi)\{t_0 := t\}))\\ &= \forall q(q(t_0) \vee \neg\forall t(\neg q(t) \rightarrow \neg\mathbf{MSOL}(\varphi)\{t_0 := t\}))\\ &= \forall q(\forall t(\mathbf{MSOL}(\varphi)\{t_0 := t\} \rightarrow q(t)) \rightarrow q(t_0)).\end{aligned}$$

In this chapter, we use ν as basic operator and μ as a defined operator, since the semantics of ν is a restricted *existential* quantification on sets of points, and μ is a restricted *universal* second order quantifier. However, $(\varphi\, \mathbf{U}^{+}\, \psi)$, which is defined by an existential first order clause, is often associated with a μ-formula: when interpreting $\mu\mathbf{TL}$ on natural models, we use the operator $\mathbf{X}$ for the unique diamond operator $\langle R\rangle$. With this notation, Lemma 3.3 can be reformulated as follows.

3.4. COROLLARY. *For any natural model $\mathcal{M}$,*

$$\mathcal{M} \models (\varphi\, \mathbf{U}^{+}\, \psi) \quad \textit{iff} \quad \mathcal{M} \models \mu q\ \ \mathbf{X}(\psi \vee \varphi \wedge q)$$

PROOF: With Lemma 3.3, the equivalence follows almost immediately from the definitions.

$$
\begin{aligned}
\mathbf{MSOL}(\mu q\, \mathbf{X}(\psi \vee \varphi \wedge q)) &= \forall q(\forall t(\mathbf{MSOL}(\mathbf{X}(\psi \vee \varphi \wedge q))\{t_0 := t\} \to q(t)) \to q(t_0)) \\
&= \mathbf{MSOL}(\forall q(\mathbf{G}^{\pm}(\mathbf{X}(\psi \vee \varphi \wedge q) \to q) \to q)) \\
&\leftrightarrow \mathbf{FOL}((\varphi\, \mathbf{U}^{+}\, \psi)) \quad \text{(according to Lemma 3.3)} \qquad \square
\end{aligned}
$$

Corollary 3.4 does not hold for more general Kripke models. In natural models, other operators can be characterized by similar $\mu\mathbf{TL}$ formulas:

$$
\begin{aligned}
\mathcal{M} \models \mathbf{F}^{+}\psi &\quad \text{iff} \quad \mathcal{M} \models \mu q\ \ \mathbf{X}(\psi \vee q) \\
\mathcal{M} \models (\varphi\, \mathbf{W}^{+}\, \psi) &\quad \text{iff} \quad \mathcal{M} \models \nu q\ \ \cancel{\mathbf{X}}(\psi \vee \varphi \wedge q) \\
\mathcal{M} \models \mathbf{G}^{*}\psi &\quad \text{iff} \quad \mathcal{M} \models \nu q\ (\psi \wedge \cancel{\mathbf{X}}\, q) \\
\mathcal{M} \models (\varphi\, \mathbf{U}^{*}\, \psi) &\quad \text{iff} \quad \mathcal{M} \models \mu q\ (\psi \vee \varphi \wedge \mathbf{X}\, q)
\end{aligned}
$$

Similarly, on tree models all **CTL** operators can be defined by $\mu\mathbf{TL}$ formulas. The same holds for most other programming logics which can be found in the literature. A formal justification of this statement will be given below in Theorem 5.10.

For certain formulas, an alternative semantical description of the ν and μ quantifiers in terms of greatest and least fixed points can be given. A function $f : 2^U \to 2^U$ is called *monotonic*, if $P \subseteq Q$ implies that $f(P) \subseteq f(Q)$. A set $Q \subseteq U$ is called a *fixed point* of f, if $Q = f(Q)$.

Let $\mathit{gfp}(f) = \bigcup\{Q \mid Q \subseteq f(Q)\}$ and $\mathit{lfp}(f) = \bigcap\{Q \mid f(Q) \subseteq Q\}$. The Knaster-Tarski fixpoint theorem [Tarski 1955] states that if f is monotonic, then $\mathit{gfp}(f)$ and $\mathit{lfp}(f)$ are the *greatest* and *least fixed point* of f.

3.5. Theorem (Knaster-Tarski). *Let $f : 2^U \to 2^U$ be monotonic. Then*
(a) $\mathit{gfp}(f) = f(\mathit{gfp}(f))$ and $\mathit{lfp}(f) = f(\mathit{lfp}(f))$, and
(b) If $Q = f(Q)$, then $Q \subseteq \mathit{gfp}(f)$ and $\mathit{lfp}(f) \subseteq Q$.

Proof: Since *gfp* and *lfp* are dual, it suffices to prove the theorem for *gfp*.

If $Q = f(Q)$, then $Q \subseteq f(Q)$. If $Q \subseteq f(Q)$, then $Q \in \{Q \mid Q \subseteq f(Q)\}$, that is, $Q \subseteq \bigcup\{Q \mid Q \subseteq f(Q)\} = \mathit{gfp}(f)$. This proves (b). Furthermore, since f is monotonic, it implies that $f(Q) \subseteq f(\mathit{gfp}(f))$. Hence for each Q, if $Q \subseteq f(Q)$ then $Q \subseteq f(\mathit{gfp}(f))$ by transitivity of set inclusion. Since each individual Q is a subset of $f(\mathit{gfp}(f))$, this means that $\bigcup\{Q \mid Q \subseteq f(Q)\} \subseteq f(\mathit{gfp}(f))$, i.e., $\mathit{gfp}(f) \subseteq f(\mathit{gfp}(f))$. This is one part of (a). Now, we use this result to infer the converse inclusion of (a): since f is monotonic, $f(\mathit{gfp}(f)) \subseteq f(f(\mathit{gfp}(f)))$. Thus, $f(\mathit{gfp}(f)) \in \{Q \mid Q \subseteq f(Q)\}$, which means $f(\mathit{gfp}(f)) \subseteq \bigcup\{Q \mid Q \subseteq f(Q)\}$. Therefore, $f(\mathit{gfp}(f)) \subseteq \mathit{gfp}(f)$. $\square$

In fact, this proof shows that the second part of the theorem can be strengthened.

3.6. Corollary. *If $f : 2^U \to 2^U$ is monotonic, then*
- *$Q \subseteq f(Q)$ implies $Q \subseteq \mathit{gfp}(f)$, and*
- *$f(Q) \subseteq Q$ implies $\mathit{lfp}(f) \subseteq Q$.*

For a more detailed discussion of other fixpoint theorems, see [Davey and Priestley 1990, Gunter and Scott 1990].

In a frame $\mathcal{F} = (U, \mathcal{I})$, any formula φ defines a set $\varphi^{\mathcal{F}} \subseteq U$ of points in the universe, namely $\varphi^{\mathcal{F}} \triangleq \{w \mid (U, \mathcal{I}, w) \models \varphi\}$. Likewise, a formula φ with a free proposition variable q defines a function $\varphi_q^{\mathcal{F}} : U \to U$ from sets of points to sets of points (a *predicate transformer*): if $Q \subseteq U$, then $\varphi_q^{\mathcal{F}}(Q) \triangleq \{w \mid (U, \mathcal{I}', w) \models \varphi\}$, where $\mathcal{I}'$ differs from $\mathcal{I}$ only in $\mathcal{I}'(q) \triangleq Q$.

3.7. Lemma. *$(\nu q\ \varphi)^{\mathcal{F}} = gfp(\varphi_q^{\mathcal{F}})$ and $(\mu q\ \varphi)^{\mathcal{F}} = lfp(\varphi_q^{\mathcal{F}})$.*

Proof: According to the definitions, $w \in gfp(\varphi_q^{\mathcal{F}})$ iff $w \in \bigcup\{Q \mid Q \subseteq \varphi_q^{\mathcal{F}}(Q)\}$, that is, if there is some $Q \subseteq U$ such that $w \in Q$ and $Q \subseteq \varphi_q^{\mathcal{F}}(Q)$. In **MSOL** this condition can be denoted as $w \models \exists q(q(t_0) \wedge \forall t(q(t) \to \mathbf{MSOL}(\varphi)\{t_0 := t\}))$. This clause is exactly the semantical translation $\mathbf{MSOL}(\nu q\ \varphi$; thus $w \in gfp(\varphi_q^{\mathcal{F}})$ iff $w \models \nu q\ \varphi$. For $lfp(\varphi_q^{\mathcal{F}})$, the dual proof holds. □

We say that a formula φ is *monotonic in q*, if the corresponding predicate transformer $\varphi_q^{\mathcal{F}}$ is monotonic. In other words, φ is monotonic in q iff $(\psi_1 \to \psi_2) \models (\varphi\{q := \psi_1\} \to \varphi\{q := \psi_2\})$ holds. φ is *monotonic*, if for each sub-formula $\nu q\ \psi$, the formula ψ is monotonic in q. Call an occurrence of a proposition variable q in a formula φ *positive* or *negative*, if it is under an even or odd number of negations. Formally, this notion is defined recursively: q is positive in the formula q. An occurrence of q in the formula $(\varphi \to \psi)$ is positive, if it is a negative occurrence in φ or a positive occurrence in ψ, and negative, if it is a positive occurrence in φ or a negative occurrence in ψ. An occurrence of q in $\langle R\rangle\varphi$ and $\nu q'\ \varphi$ is positive or negative, if it is positive or negative in φ, respectively. A formula φ is called *positive in q*, if every free occurrence of q in φ is positive. It is *positive*, if each sub-formula $\nu q\ \psi$ is positive in q.

3.8. Lemma. *If φ is positive in q, then $\varphi_q^{\mathcal{F}}$ is a monotonic predicate transformer.*

Proof: This statement can be proved by induction on the structure of φ. The induction basis, namely formulas which are atomic propositions, proposition variables or boolean constants, is immediate. For the inductive step, assume that $P \subseteq Q$. If $(\varphi \to \psi)$ is positive in q, then ψ must be positive and φ must be negative in q. Therefore, $\neg\varphi$ is positive in q. The induction hypothesis is that $\psi_q^{\mathcal{F}}(P) \subseteq \psi_q^{\mathcal{F}}(Q)$ and $\neg\varphi_q^{\mathcal{F}}(P) \subseteq \neg\varphi_q^{\mathcal{F}}(Q)$. From this we can infer that $\varphi_q^{\mathcal{F}}(Q) \subseteq \varphi_q^{\mathcal{F}}(P)$. Therefore, if $\varphi_q^{\mathcal{F}}(P) \subseteq \psi_q^{\mathcal{F}}(P)$ then $\varphi_q^{\mathcal{F}}(Q) \subseteq \psi_q^{\mathcal{F}}(Q)$. This follows from $\varphi_q^{\mathcal{F}}(Q) \subseteq \varphi_q^{\mathcal{F}}(P) \subseteq \psi_q^{\mathcal{F}}(P) \subseteq \psi_q^{\mathcal{F}}(Q)$. In other words, $(\varphi \to \psi)_q^{\mathcal{F}}(P) \subseteq (\varphi \to \psi)_q^{\mathcal{F}}(Q)$. For the case $\langle R\rangle\varphi$, the induction hypothesis is that $\varphi_q^{\mathcal{F}}(P) \subseteq \varphi_q^{\mathcal{F}}(Q)$. Then, $\{w \mid \exists w'(w, w') \in \mathcal{I}(R) \wedge w' \in \varphi_q^{\mathcal{F}}(P)\} \subseteq \{w \mid \exists w'(w, w') \in \mathcal{I}(R) \wedge w' \in \varphi_q^{\mathcal{F}}(Q)\}$. In other words, $(\langle R\rangle\varphi)_q^{\mathcal{F}}(P) \subseteq (\langle R\rangle\varphi)_q^{\mathcal{F}}(Q)$. Similarly, for formulas $\nu q'\varphi$, where q and q' are different variables, the induction hypothesis is that $\varphi_{q,q'}^{\mathcal{F}}(P, X) \subseteq \varphi_{q,q'}^{\mathcal{F}}(Q, X)$ for all X. Therefore, $X \subseteq (\varphi)_{q,q'}^{\mathcal{F}}(P, X)$ implies $X \subseteq (\varphi)_{q,q'}^{\mathcal{F}}(Q, X)$ for all X. Consequently, $\{w \mid \text{for some } X,\ w \in X \text{ and } X \subseteq \varphi_{q,q'}^{\mathcal{F}}(P, X)\} \subseteq \{w \mid \text{for some } X,\ w \in X \text{ and } X \subseteq \varphi_{q,q'}^{\mathcal{F}}(Q, X)\}$. According to the definition, this is the semantics of

$(\nu q'\varphi)_q^{\mathcal{F}}(P) \subseteq (\nu q'\varphi)_q^{\mathcal{F}}(Q)$. The last case is $\nu q \varphi$. Since this formula has no free occurrence of variable q, its denotation $(\nu q \varphi)_q^{\mathcal{F}}$ is a constant function. Trivially, constant functions are monotonic. □

The converse of this statement does not hold in general. In particular, [Ajtai and Gurevich 1987] shows that there is a formula which is monotonic on all finite structures but has no positive equivalent.

3.9. Corollary. *If φ is positive, then*

- $\models (\nu q\ \varphi \leftrightarrow \varphi\{q := \nu q\ \varphi\})$ *and* $\models (\mu q\ \varphi \leftrightarrow \varphi\{q := \mu q\ \varphi\})$.
- *If* $(U, \mathcal{I}) \models (\chi \leftrightarrow \varphi\{q := \chi\})$ *then both* $(U, \mathcal{I}) \models (\chi \to \nu q\ \varphi)$ *and* $(U, \mathcal{I}) \models (\mu q\ \varphi \to \chi)$.
- $(U, \mathcal{I}) \models (\chi \to \varphi\{q := \chi\})$ *implies* $(U, \mathcal{I}) \models (\chi \to \nu q\ \varphi)$, *and* $(U, \mathcal{I}) \models (\varphi\{q := \chi\} \to \chi)$ *implies* $(U, \mathcal{I}) \models (\mu q\ \varphi \to \chi)$

Proof: If φ is positive in q, then $\varphi_q^{\mathcal{F}}$ is monotonic according to Lemma 3.8. Theorem 3.5 asserts that $\mathit{gfp}(\varphi_q^{\mathcal{F}}) = \varphi_q^{\mathcal{F}}(\mathit{gfp}(\varphi_q^{\mathcal{F}}))$. In the notation of Lemma 3.7, this means $(\nu q\ \varphi)^{\mathcal{F}} = \varphi_q^{\mathcal{F}}((\nu q\ \varphi)^{\mathcal{F}})$. Moreover, $\varphi_q^{\mathcal{F}}((\nu q\ \varphi)^{\mathcal{F}}) = (\varphi_q\{q := \nu q\ \varphi\})^{\mathcal{F}}$. Therefore, $\mathcal{M} \models (\nu q\ \varphi \leftrightarrow \varphi_q\{q := \nu q\ \varphi\})$. The other statements are shown similarly. □

According to Corollary 3.4, $(\varphi\, \mathbf{U}^+\, \psi)$ and $(\varphi\, \mathbf{W}^+\, \psi)$ in natural models are least and greatest fixed points of $\mathbf{X}(\psi \vee \varphi \wedge q)$ and $\cancel{\mathbf{X}}(\psi \vee \varphi \wedge q)$, respectively. Therefore, the following *recursion* and *induction* axioms hold:

- $\models (\varphi\, \mathbf{U}^+\, \psi) \leftrightarrow \mathbf{X}(\psi \vee \varphi \wedge (\varphi\, \mathbf{U}^+\, \psi))$ and $\models (\varphi\, \mathbf{W}^+\, \psi) \leftrightarrow \cancel{\mathbf{X}}(\psi \vee \varphi \wedge (\varphi\, \mathbf{W}^+\, \psi))$.
- $(U, \mathcal{I}) \models (\mathbf{X}(\psi \vee \varphi \wedge \chi) \to \chi)$ implies $(U, \mathcal{I}) \models ((\varphi\, \mathbf{U}^+\, \psi) \to \chi)$, and $(U, \mathcal{I}) \models (\chi \to \cancel{\mathbf{X}}(\psi \vee \varphi \wedge \chi))$ implies $(U, \mathcal{I}) \models (\chi \to (\varphi\, \mathbf{W}^+\, \psi))$.

In particular, for $\mathbf{F}^+$, $\mathbf{G}^+$, $\mathbf{F}^*$ and $\mathbf{G}^*$, we have

- $\models (\mathbf{F}^+\, \psi) \leftrightarrow \mathbf{X}(\psi \vee \mathbf{F}^+\, \psi)$ and $\models (\mathbf{G}^+\, \varphi) \leftrightarrow \cancel{\mathbf{X}}(\varphi \wedge \mathbf{G}^+\, \varphi)$.
- $\models (\mathbf{F}^*\, \psi) \leftrightarrow (\psi \vee \mathbf{X}\, \mathbf{F}^*\, \psi)$ and $\models (\mathbf{G}^*\, \varphi) \leftrightarrow (\varphi \wedge \cancel{\mathbf{X}}\, \mathbf{G}^*\, \varphi)$.
- $(U, \mathcal{I}) \models (\mathbf{X}(\psi \vee \chi) \to \chi)$ implies $(U, \mathcal{I}) \models ((\mathbf{F}^+\, \psi) \to \chi)$, and $(U, \mathcal{I}) \models (\chi \to \cancel{\mathbf{X}}(\varphi \wedge \chi))$ implies $(U, \mathcal{I}) \models (\chi \to \mathbf{G}^+\, \varphi)$.
- $(U, \mathcal{I}) \models ((\psi \vee \mathbf{X}\, \chi) \to \chi)$ implies $(U, \mathcal{I}) \models ((\mathbf{F}^*\, \psi) \to \chi)$, and $(U, \mathcal{I}) \models (\chi \to (\varphi \wedge \cancel{\mathbf{X}}\, \chi))$ implies $(U, \mathcal{I}) \models (\chi \to \mathbf{G}^*\, \varphi)$.

As we have shown, positive $\mu\mathbf{TL}$ formulas denote greatest or least fixed points of predicate transformers. For nonmonotonic formulas, the existence of fixed points is not granted. For example, there is no $Q \subseteq U$ satisfying $Q = U \setminus Q$; thus, there is no fixed point of $(\neg q)_q^{\mathcal{F}}$. However, the **MSOL** semantics of $\nu q\ \neg q$ is $\exists q(q(t_0) \wedge \forall t(q(t) \to \neg q(t)))$, which is equivalent to the well-defined value $\bot$. On general Kripke-models, positive $\mu\mathbf{TL}$ is strictly weaker in expressiveness than unrestricted $\mu\mathbf{TL}$. Even unrestricted $\mu\mathbf{TL}$ can, in turn, express fewer properties of Kripke models than monadic second order logic:

3.10. LEMMA. *Consider the class of all Kripke models.*

(a) *There is no positive* μ**TL** *formula which is equivalent to* $\nu q(\langle R \rangle \neg q)$.

(b) *There is no* μ**TL** *formula which is equivalent to* $\forall t\mathtt{p}(t)$

PROOF: For (a), consider $\varphi \triangleq \nu q(\langle R \rangle \neg q)$. Then $\mathbf{MSOL}(\varphi) = \exists q(q(t_0) \wedge \forall t(q(t) \rightarrow \exists t'(tRt' \wedge \neg q(t'))))$. This formula is equivalent to the first order condition $\exists t(t_0 Rt \wedge t_0 \neq t)$: in one direction, if there is some q such that $w \in \mathcal{I}(q)$ and $w \models \forall t(q(t) \rightarrow \exists t'(tRt' \wedge \neg q(t')))$, then there must be a point reachable from w which is not in $\mathcal{I}(q)$, i.e., different from w. For the reverse implication, assume that $w \models \exists t(t_0 Rt \wedge t_0 \neq t)$ and let $\mathcal{I}(q) \triangleq \{w\}$. Then $w \models q(t_0)$ and $w \models \forall t(q(t) \rightarrow \exists t'(tRt' \wedge \neg q(t')))$. Therefore, $w \models \varphi$.

There is no positive formula which can express this property: consider the frame $\mathcal{F} \triangleq (U, \mathcal{I})$, where $U \triangleq \{w_0, w_1\}$, $\mathcal{I}(R) \triangleq \{(w_0, w_0), (w_0, w_1), (w_1, w_1)\}$ and $\mathcal{I}(\mathtt{p}) = \{\}$ for all $\mathtt{p} \in \mathcal{P}$. Then $w_0 \models \varphi$ and $w_1 \not\models \varphi$. For each positive formula ψ, however, it holds that $w_0 \models \psi$ iff $w_1 \models \psi$. To prove this, we show by induction on the structure of ψ that $\psi^{\mathcal{F}} = \{\}$ or $\psi^{\mathcal{F}} = U$. For propositional formulas, this is immediate; the case $\langle R \rangle \psi$ follows from the definition of $\mathcal{F}$. The only remaining case are formulas $\nu q\psi$. According to the induction hypothesis, either $(\psi\{q := \top\})^{\mathcal{F}} = \{\}$ or $(\psi\{q := \top\})^{\mathcal{F}} = U$. In the first case, from the fact that $(\nu q\psi)^{\mathcal{F}} \subseteq \top^{\mathcal{F}}$ and monotonicity of ψ we infer that $(\psi\{q := \nu q\psi\})^{\mathcal{F}} \subseteq (\psi\{q := \top\})^{\mathcal{F}} = \{\}$. The first part of Theorem 3.5 implies that $(\nu q\psi)^{\mathcal{F}} \subseteq (\psi\{q := \nu q\psi\})^{\mathcal{F}}$; therefore, $(\nu q\psi)^{\mathcal{F}} = \{\}$. In the second case, $U = \top^{\mathcal{F}} = (\psi\{q := \top\})^{\mathcal{F}}$. With the second part of Theorem 3.5, it follows that $\top^{\mathcal{F}} \subseteq (\nu q\psi)^{\mathcal{F}}$, i.e., $(\nu q\psi)^{\mathcal{F}} = U$.

Statement (b) holds since the truth of μ**TL** formulas is preserved under disjoint unions of models, whereas $\varphi \triangleq \forall t\mathtt{p}(t)$ can be invalidated by adding an isolated point w with $w \not\models \mathtt{p}$. Formally, consider the models $\mathcal{M}_0 \triangleq (U_0, \mathcal{I}, w_0)$ and $\mathcal{M}_1 \triangleq (U_1, \mathcal{I}, w_0)$, where $U_0 \triangleq \{w_0\}$, $U_1 \triangleq \{w_0, w\}$, $\mathcal{I}(R) \triangleq \{\}$ and $\mathcal{I}(\mathtt{p}) \triangleq \mathcal{I}(q) \triangleq \{w_0\}$. Then $\mathcal{M}_0 \models \varphi$ and $\mathcal{M}_1 \not\models \varphi$, whereas for each μ**TL** formula ψ it holds that $\mathcal{M}_0 \models \varphi$ iff $\mathcal{M}_1 \models \varphi$. As above, the only interesting case is $\nu q\psi$. If $\mathcal{M}_0 \models \nu q\psi$ then $w_0 \models \psi$, which implies $\mathcal{M}_1 \models \nu q\psi$. In the other direction, $\mathcal{M}_1 \models \nu q\psi$ implies that either $\mathcal{M}_1 \models \psi$ or $\mathcal{M}_1' \triangleq (U_1, \mathcal{I}_1, w_0) \models \psi$, where $\mathcal{I}_1(q) = U_1$. In the first case, $\mathcal{M}_0 \models \nu q\psi$ follows directly. In the second case, $\mathcal{M}_1 \models \psi\{q := \top\}$, which implies $\mathcal{M}_0 \models \psi\{q := \top\}$ by the induction hypothesis. From this, it follows that $\mathcal{M}_0 \models \nu q\psi$. ☐

If the model is *connected* (that is, $\forall w, w'(w < w' \vee w = w' \vee w > w')$), then every point is reachable from the current point. In this case, the operator $\mathbf{G}^{\pm}$ (the *universal modality*) can replace the first-order universal quantifier: $\mathcal{M} \models \forall t\mathtt{p}(t)$ iff $\mathcal{M} \models \mathbf{G}^{\pm}\,\mathtt{p}$. In this case,

$$\mathcal{M} \models \nu q\, \varphi \quad \text{iff} \quad \mathcal{M} \models \exists q(q \wedge \mathbf{G}^{\pm}(q \rightarrow \varphi)),$$
$$\mathcal{M} \models \mu q\, \varphi \quad \text{iff} \quad \mathcal{M} \models \forall q(\mathbf{G}^{\pm}(\varphi \rightarrow q) \rightarrow q).$$

Hence, on connected models (and, in particular, on natural models) μ**TL** is at most as expressive as **qTL** (and **MSOL**). Since μ**TL** does not contain any past-operators, there is no μ**TL** formula which is equivalent to $\mathbf{F}^{-}\,\top$. Subsequently, however, we will show that for *initial validity* in natural models a translation from **qTL** (or **MSOL**) into positive μ**TL** exists. Since the proof uses ω-*regular languages* and ω-*automata*, it is postponed to subsection 3.4.

3.3. ω-automata and ω-languages

Given a (finite or infinite) natural model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$, the interpretation $\mathcal{I}$ defines a mapping $\mathcal{I} : \mathcal{P} \to 2^U$ from propositions into subsets of the universe. Define a *labelling function* $\mathcal{L} : U \to 2^{\mathcal{P}}$ by

$$\mathtt{p} \in \mathcal{L}(w) \qquad \text{iff} \qquad w \in \mathcal{I}(\mathtt{p})$$

That is, $\mathcal{L}(w) \triangleq \{\mathtt{p} \mid w \in \mathcal{I}(\mathtt{p})\}$ is the *label* of point $w \in U$. If $U = (w_0, w_1, w_2, ...)$, then the sequence $\sigma = (\mathcal{L}(w_0), \mathcal{L}(w_1), \mathcal{L}(w_2), ...)$ is called the ω-*word* of $\mathcal{M}$ over the *alphabet* $\Sigma \triangleq 2^{\mathcal{P}}$. A set of ω-words is called an ω-*language*.

Let $\mathcal{F} \triangleq (U, \mathcal{I})$ be the frame of a natural model. Formula φ is *initially valid* in $\mathcal{F}$, if $(U, \mathcal{I}, w_0) \models \varphi$, where w_0 is the unique initial point of U (which has no predecessors). For any such frame $\mathcal{F}$ it holds that φ is universally valid iff $\mathbf{G}^* \varphi$ is initially valid, and φ is initially valid iff $(\mathbf{G}^- \perp \to \varphi)$ is universally valid.

We say that a linear-time logic formula *defines* the set of all natural frames in which it is initially valid. Thus every such formula defines the ω-language given by these frames. We now show that in order to define languages by formulas it suffices to restrict attention to the future fragment of temporal logic. The separation Lemma 2.6 states that any **LTL**-formula can be separated into a boolean combination of pure future, pure present and pure past formulas. It can be extended to **qTL**:

3.11. LEMMA. **qTL** *has the separation property on natural models.*

PROOF: Note that the formula $\exists q(\varphi \vee \psi) \leftrightarrow (\exists q\ \varphi \vee \exists q\ \psi)$ is valid. Moreover, if φ_1,...,φ_n are pure past, ψ_1,...,ψ_m are pure present and χ_1,...,χ_l are pure future, then $\exists q(\bigwedge \varphi_i \wedge \bigwedge \psi_j \wedge \bigwedge \chi_k)$ is equivalent to $(\exists q \bigwedge \varphi_i \wedge \exists q \bigwedge \psi_j \wedge \exists q \bigwedge \chi_k)$. Informally, this can be seen as follows: $\exists q(\varphi \wedge \psi) \to (\exists q \varphi \wedge \exists q \psi)$ is a tautology. In the other direction, assume that the past-formulas φ_1,...,φ_n are valid in the model $(U, \mathcal{I}, w_0)$ where $\mathcal{I}(q) \triangleq Q_1$, the present-formulas ψ_j are valid with $\mathcal{I}(q) \triangleq Q_2$, and the future-formulas χ_k are valid if $\mathcal{I}(q) \triangleq Q_3$, then the conjunction of past, present and future part is valid if $\mathcal{I}(q) \triangleq (\{w \mid w < w_0\} \cap Q_1) \cup (w_0 \cap Q_2) \cup (\{w \mid w > w_0\} \cap Q_3)$.

Now assume that $\varphi \triangleq \exists q \psi$, and show that there is an equivalent separated formula. The induction hypothesis is that for ψ there exists an equivalent formula ψ' which is a boolean combination of pure future, past and present formulas. Let $\psi'' \triangleq \bigvee \bigwedge (\varphi_i \wedge \psi_j \wedge \chi_k)$ be ψ' in disjunctive normal form, where all φ_i are pure past, ψ_j pure present and χ_k pure future. Applying the above formulas we see that $\exists q \psi'' \leftrightarrow \bigvee \bigwedge (\exists q \varphi_i \wedge \exists q \psi_j \wedge \exists q \chi_k)$. This formula is separated and equivalent to φ.$\square$

3.12. LEMMA. *For any* **LTL** *or* **qTL** *formula* φ *there exists an* **LTL** *or* **qTL** *future formula (without* $\mathbf{U}^{-}$*-operators) defining the same language.*

PROOF: Given a separated formula φ, let φ^{+} be the formula φ where every subformula $(\varphi\,\mathbf{U}^{-}\,\psi)$ is replaced by $\perp$. Then φ is initially valid in any natural model $\mathcal{M}$ iff ψ^{+} is initially valid in $\mathcal{M}$. Thus φ and φ^{+} define the same language. ☐

Languages can also be defined by *(ω-)regular expressions* and by *finite (ω-) automata*.

The language of *(ω-)regular expression* is defined similar to the language of usual regular expressions, with an additional operation denoting infinite repetition of a subexpression.

- Every letter from the alphabet is an ω-regular expression.
- If α and β are ω-regular expressions, then so are ε, $(\alpha+\beta)$, $(\alpha;\beta)$ and α^{+}.
- If α is an ω-regular expression, then so is α^{ω}.

Every ω-regular expression defines an ω-language: the letter $a \subseteq \mathcal{P}$ defines $\{(a)\}$, i.e., a one-word language (one-element set) consisting of a one-letter word (one-element sequence). ε denotes the empty language, and $(\alpha+\beta)$, $(\alpha;\beta)$ and α^{+} denote union, sequential composition and finite iteration of languages. α^{ω} denotes the language of all words consisting of an infinite concatenation of words from α. A language is called ω-regular if it can be defined by an ω-regular expression.

We use boolean terms over $\mathcal{P}$ to denote (unions of) letters. For example, if $\mathcal{P} = \{\mathtt{p1},\mathtt{p2}\}$ then $(\neg\mathtt{p1}\wedge\mathtt{p2})$ denotes the letter $\{\mathtt{p2}\}$, and $(\neg\mathtt{p1}\vee\mathtt{p2})$ denotes $\{\}+\{\mathtt{p2}\}+\{\mathtt{p1},\mathtt{p2}\}$.

As an example for an ω-regular expression, consider $(\neg\mathtt{p1})^{\omega} + (\top^{+};\mathtt{p2})^{\omega}$. This expression defines the set of all infinite words $(\sigma_0,\sigma_1,\sigma_2,...)$ such that either for all i it holds that $\mathtt{p1} \notin \sigma_i$, or for infinitely many i it holds that $\mathtt{p2} \in \sigma_i$. That is, it defines the set of natural models $\mathcal{M}$ such that $\mathcal{M} \models \mathbf{G}^{*}(\neg\mathtt{p1} \wedge \mathbf{X}\,\top) \vee \mathbf{G}^{*}\,\mathbf{F}^{+}\,\mathtt{p2}$. Since this formula implies $\mathbf{G}^{*}\,\mathbf{X}\,\top$, each of its natural models must be infinite.

An *ω-automaton* or *fair transition system* over the alphabet $\Sigma = 2^{\mathcal{P}}$ is defined like a usual (nondeterministic) automaton with an additional recurrence set ("fairness constraint"); it is a tuple $(S,\Delta,S_0,S_{acc},S_{rec})$, where

- S is a set of states,
- $\Delta \subseteq S\times\Sigma\times S$ is the transition relation,
- $S_0 \subseteq S$ is the set of initial states,
- $S_{acc} \subseteq S$ is the set of accepting states (for finite words), and
- $S_{rec} \subseteq S$ is the set of recurring states (for infinite words).

A *Büchi-automaton* is a finite ω-automaton, that is, a fair transition system where the set S of states is finite. A *transition system* (or *labelled transition system*) is a fair transition system where $S_{acc} = S_{rec} = S$. A *weakly fair transition system* is an ω-automaton where $S_{rec} = S$ and $S_{acc} = \{s \mid \forall a, s'(s,a,s') \notin \Delta\}$. That is, in a weakly fair transition system all states are recurring, and states are accepting iff they are terminal. Usually, when talking about labelled and weakly fair transition systems, we omit the redundant components S_{acc} and S_{rec}.

A (finite or infinite) nonempty word $\sigma \triangleq (\sigma_0, \sigma_1, ...)$ is *accepted* by an automaton $(S, \Delta, S_0, S_{acc}, S_{rec})$, if there is a function ρ assigning to any $i < |\sigma|$ a state $\rho(\sigma_i) \in S$ of the automaton such that

- $\rho(0) \in S_0$,
- For all $0 \leq i < n$, $(\rho(i), \sigma_i, \rho(i+1)) \in \Delta$, and
- $(\rho(n), \sigma_n, s) \in \Delta$ for some $s \in S_{acc}$, if σ is finite with last letter w_n, and
- $\mathit{inf}(\rho) \cap S_{rec} \neq \{\}$, if σ is infinite, where $\mathit{inf}(\rho)$ is the set of states that appear infinitely often in the range of ρ. That is, at least one recurring state must be selected infinitely often.

For alternative acceptance conditions, see [Thomas 1990][4]. We say that an automaton accepts a natural model $\mathcal{M}$, if it accepts the ω-word of $\mathcal{M}$. The language of a transition system consists of all paths through the transition graph; this language is prefix-closed (for any word in the language, all of its prefixes are also contained). The language defined by a weakly fair transition system consists of all maximal paths through the graph.

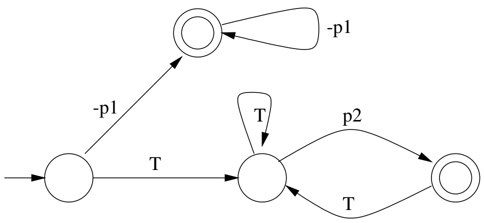


Figure 3: A Büchi automaton accepting $(\neg\mathtt{p1})^\omega + (\top^+;\mathtt{p2})^\omega$

As an example of a Büchi-automaton, consider Figure 3. This automaton accepts (i.e., defines) exactly the same language as the example ω-regular expression above. In general, for any ω-regular expression we can construct such a Büchi-automaton and vice versa; Büchi-automata can define all and only ω-regular languages.

3.13. LEMMA. *ω-regular expressions and Büchi-automata are of equal expressive power.*

PROOF: The proof of this statement is similar as for automata on finite words: for one direction, we have to show that the Büchi acceptance condition can be captured

[4]In the literature, a fairness constraint in a transition systems is sometimes defined to be a set of pairs (s, e), where $s \in S$ is a state and $e \in \Delta$ is an edge. It imposes the condition that if s appears infinitely often, then e must be taken infinitely often in each accepted word. It can be shown that for our purposes these definitions are equivalent. See [Anuchitanukul 1995] for the relationships and translations between these two notions for various acceptance conditions.

by an appropriate regular expression. Let $L(s_i, s_j)$ be a regular expression for the language of finite nonempty words sending an automaton from state s_i into state s_j. Then the ω-regular expression associated with any Büchi-automaton is

$$\Sigma\{L(s_0, s) \mid s_0 \in S_0,\ s \in S_{acc}\} + \Sigma\{L(s_0, s); L(s, s)^\omega \mid s_0 \in S_0,\ s \in S_{rec}\}$$

For the other direction it must be shown that Büchi-automata are closed under single letters, the empty language, union, concatenation, and finite and infinite repetition. All of these constructions are straightforward extensions of the appropriate constructions for automata on finite words [Hopcroft and Ullman 1979]. □

The automaton resulting from this proof is highly nondeterministic. An automaton is called *deterministic*, if its transition relation is a function $\Delta : S \times \Sigma \rightarrow S$. For each nondeterministic finite automaton on finite words an equivalent deterministic one is given by the well known powerset construction of Rabin and Scott [Hopcroft and Ullman 1979]. The same holds for finite transition systems. In contrast, for nondeterministic Büchi-automata it is not always possible to construct an equivalent deterministic one. For example, consider the language $\mathcal{L}$ of all words containing only finitely many $\mathtt{p}$. This language is defined by the formula $\mathbf{F}^* \mathbf{G}^+ \neg\mathtt{p}$ or the ω-regular expression $(\top^+ + (\top^+; \neg\mathtt{p}^\omega))$. However, there is no deterministic Büchi-automaton defining $\mathcal{L}$: Assume for contradiction that $\mathcal{L}$ is the language of $\mathcal{A}$. Then $\mathcal{A}$ must accept $(\sigma; (\neg\mathtt{p})^\omega)$ for any finite word σ. In particular, from any reachable state some recurring state is reached by a finite number of $\neg\mathtt{p}$-transitions. Let m be the maximum of these numbers. Therefore, in the run of $\mathcal{A}$ on the word $(\mathtt{p}; (\neg\mathtt{p})^m)^\omega$ infinitely often recurring states are visited. Thus, this word also is accepted by $\mathcal{A}$. This is a contradiction, since it is not in $\mathcal{L}$.

3.4. Automata and Logics

Büchi [Büchi 1962] showed that his automata are closed under complement; this is a highly nontrivial proof. The best known construction for complementing Büchi-automata was given in [Safra 1988]; it involves an exponential blowup of the number of states of the automaton. More precisely, if A has n states, then it can be shown that the smallest automaton accepting the complement language of A in general has $O(2^{n \log n})$ states. For more information on the complementation problem for Büchi automata, see [Sistla, Vardi and Wolper 1987, Thomas 1990].

Closure under complement can be used to show that Büchi-automata are at least as expressive as **qTL**.

3.14. LEMMA. *For every* **qTL** *formula there is a Büchi-automaton defining the same language.*

PROOF: According to Lemma 3.12 it suffices to give a translation for formulas without $\mathbf{U}^-$. An automaton for the proposition $\mathtt{p} \in \mathcal{P}$ is given by the trivial two-state machine $(\{s_0, s_{acc}\}, \Delta, \{s_0\}, \{s_{acc}\}, \{\})$, where $(s_0, a, s_{acc}) \in \Delta$ iff $\mathtt{p} \in a$. An automaton for $\bot$ is one which never accepts. From an automaton for φ, an automaton for $\mathbf{X}\,\varphi$ and $\mathbf{F}^{+}\,\varphi$ can be built by an appropriate prefixing with a single step or loop on the initial states. According to the remark following Lemma 3.3, $\mathbf{U}^{+}$ can be expressed with $\mathbf{X}$, $\mathbf{F}^{+}$ and second order quantification. Implications $(\varphi \to \psi)$ can be written as $(\neg\varphi \vee \psi)$ and thus be reduced to unions and complements. Finally, existential second order quantification amounts to the projection of the automaton onto a smaller alphabet: given an automaton $A = (S, \Delta, S_0, S_{acc}, S_{rec})$ over the alphabet $2^{(\mathcal{P}\cup\mathcal{Q})}$ which accepts the models of φ, the automaton $A'(S, \Delta', S_0, S_{acc}, S_{rec})$ accepts all models of $\exists q\ \varphi$, where $(s_i, a, s_j) \in \Delta'$ iff $(s_i, a \setminus \{q\}, s_j) \in \Delta$ or $(s_i, a \cup \{q\}, s_j) \in \Delta'$. $\square$

In particular, since **LTL** is a sublanguage of **qTL**, for every **LTL** formula there exists a corresponding Büchi-automaton. In Section 7, we will describe a tableaux decision procedure, which can be seen as an efficient algorithm to construct a Büchi-automaton from a formula. Other aspects of the connections between temporal logics, monadic logics and automata can be found in [Thomas 1999].

We now show that ω-regular expressions are at most as expressive as $\mu\mathbf{TL}$:

3.15. LEMMA. *For every ω-regular expression there exists a $\mu\mathbf{TL}$-formula describing the same language.*

PROOF: The proof associates with every ω-regular expression φ a $\mu\mathbf{TL}$-formula $\mu\mathbf{TL}_q(\varphi)$ with at most one free proposition variable q indicating the end of the sequence.

- $\mu\mathbf{TL}_q(P) \triangleq (\bigwedge_{\mathtt{p}\in P} \mathtt{p} \wedge \bigwedge_{\mathtt{p}\notin P} \neg\mathtt{p} \wedge q)$, if $P \in 2^{\mathcal{P}}$
- $\mu\mathbf{TL}_q(\epsilon) \triangleq \bot$
- $\mu\mathbf{TL}_q(\varphi + \psi) \triangleq (\mu\mathbf{TL}_q(\varphi) \vee \mu\mathbf{TL}_q(\psi))$
- $\mu\mathbf{TL}_q(\varphi;\psi) \triangleq \mu\mathbf{TL}_q(\varphi)\{q := \mathbf{X}\,\mu\mathbf{TL}_q(\psi)\}$
- $\mu\mathbf{TL}_q(\varphi^{+}) \triangleq \mu q_1\ (\mu\mathbf{TL}_q(\varphi)\{q := q \vee \mathbf{X}\,q_1\})$
- $\mu\mathbf{TL}_q(\varphi^{\omega}) \triangleq \nu q_1\ (\mu\mathbf{TL}_q(\varphi)\{q := \mathbf{X}\,q_1\})$

If φ defines a language of infinite strings, then $\mu\mathbf{TL}_q(\varphi)$ does not contain any free occurrence of q. However, if φ defines a language of finite strings, then $\mu\mathbf{TL}_q(\varphi)$ contains the free proposition variable q denoting the final point. A finite string is characterized by the fact that in its last point the formula $\not\mathbf{X}\,\bot$ holds. Therefore, the $\mu\mathbf{TL}$-formula corresponding to an ω-regular expression φ is defined as $\mu\mathbf{TL}(\varphi) \triangleq \mu\mathbf{TL}_q(\varphi)\{q := \not\mathbf{X}\,\bot\}$. It can be shown that $\mu\mathbf{TL}(\varphi)$ defines the same language as the ω-regular expression φ. $\square$

As an example, consider the expression $(\neg\mathtt{p1})^{\omega} + (\top^{+};\mathtt{p2})^{\omega}$.

$$
\begin{aligned}
&\mu\mathbf{TL}((\neg\mathtt{p1})^{\omega} + (\top^{+};\mathtt{p2})^{\omega})\\
&\quad = \nu q_1\ (\mu\mathbf{TL}(\neg\mathtt{p1})\{q := \mathbf{X}\,q_1\}) \vee \nu q_2(\mu\mathbf{TL}(\top^{+};\mathtt{p2})\{q := \mathbf{X}\,q_2\})\\
&\quad = \nu q_1\ ((\neg\mathtt{p1} \wedge q)\{q := \mathbf{X}\,q_1\}) \vee \nu q_2((\mu\mathbf{TL}(\top^{+})\{q := \mathbf{X}\,\mu\mathbf{TL}(\mathtt{p2})\})\{q := \mathbf{X}\,q_2\})\\
&\quad = \nu q_1\ (\neg\mathtt{p1} \wedge \mathbf{X}\,q_1) \vee\\
&\qquad\quad \nu q_2(\mu q_3(\top \wedge q)\{q := q \vee \mathbf{X}\,q_3\}\{q := \mathbf{X}(\mathtt{p2} \wedge q)\}\{q := \mathbf{X}\,q_2\})
\end{aligned}
$$

$$
\begin{aligned}
&= \nu q_1 \ (\neg \mathtt{p1} \wedge \mathbf{X}\, q_1) \vee \nu q_2(\mu q_3(\mathbf{X}(\mathtt{p2} \wedge q) \vee \mathbf{X}\, q_3)\{q := \mathbf{X}\, q_2\}) \\
&= \nu q_1 \ (\neg \mathtt{p1} \wedge \mathbf{X}\, q_1) \vee \nu q_2(\mu q_3\, \mathbf{X}(\mathtt{p2} \wedge \mathbf{X}\, q_2 \vee q_3)) \\
&\leftrightarrow \nu q_1 \ (\neg \mathtt{p1} \wedge \mathbf{X} \top \wedge \cancel{\mathbf{X}}\, q_1) \vee \nu q_2(\mu q_3\, \mathbf{X}(\mathtt{p2} \vee q_3) \wedge \cancel{\mathbf{X}}\, q_2) \\
&= \mathbf{G}^{*}(\neg \mathtt{p1} \wedge \mathbf{X} \top) \vee \mathbf{G}^{*}\, \mathbf{F}^{+}\, \mathtt{p2}
\end{aligned}
$$

This lemma closes the circle in the expressiveness results of second order languages.

3.16. THEOREM (Büchi, Wolper, Sistla). *To define ω-languages, the following formalisms are of equal expressive power:*

i. μ**TL**
ii. **qTL**
iii. **MSOL**
iv. *Büchi-automata*
v. *ω-regular expressions*

PROOF: For every μ**TL**-formula there exists an equivalent **qTL**-formula by definition; on natural models **qTL** is equal in expressiveness to **MSOL** by Lemma 3.2; according to Lemma 3.14, for every **qTL** (or **MSOL**) formula there is a Büchi-automaton defining the set of its models; by Lemma 3.13, Büchi-automata are equivalent to ω-regular expressions; and these in turn can be described by μ**TL**-formulas as shown in Lemma 3.15. □

Similar results can be proved about logics with past operators on integer models (bi-infinite words) and two-way automata, and about branching time logics (μ**TL**/**qTL** on tree models) and tree automata ($\Delta \subseteq S \times 2^{\mathcal{P}} \times (\mathcal{R} \times S)^n$) (see [Niwinsky 1988, Thomas 1990, Schlingloff 1992*b*]).

4. Model Transformations and Properties

As we have seen, linear temporal formulas and ω-automata both can be used to describe sets of infinite sequences. The practical difference is, that logic tends to be more "descriptive", specifying *what* a system should do, whereas automata tend to be more "machine-oriented", indicating *how* it should be done. Logical formulas are "global", they are interpreted on the whole structure, whereas automata are "local", describing single states and transitions.

Therefore, traditionally automata or related models are used to give an abstract account of the *system* to be verified, whereas formulas are used to specify *properties* of these systems. But, since it is possible to translate between automata and formulas and back, this choice is a matter of complexity, of available algorithms and of taste. We could equally well define both system and properties in temporal logic; in this case we would have to prove an implication formula (Section 7 will explain how to do this). Another alternative is that both the implementation and the specification are given as automata, where the latter is more "abstract" than the former. Then we have to prove that one can *simulate* the other.

In the next sections, we describe various transformations between models such as simulations and refinements, and investigate the preservation of logical properties under these transformations.

4.1. Models, Automata and Transition Systems

The previous section related ω-automata and linear temporal formulas via the ω-language accepted by the automaton and the set of natural models in which the formula is initially valid. There is, however, a more direct connection on the structural level. Let $\mathcal{M} = (U, \mathcal{I}, w_0)$ be a Kripke-model with predicates from $\mathcal{P}$ and accessibility relations from $\mathcal{R}$. Consider the alphabet $\Sigma = 2^{\mathcal{P}} \times \mathcal{R}$, and let $\sigma = (\sigma_0\sigma_1\sigma_2...)$ be an ω-word, where $\sigma_i = (a_i, R_i)$. We say that σ is *generated by* $\mathcal{M}$ if there exists a mapping ρ from indices of letters of σ into points of U, such that

- $\rho(0) = w_0$,
- if $\rho(i) = w$, then $a_i = \mathcal{L}(w)$,
- if $\rho(i) = w$ and $\rho(i+1) = w'$, then $(w, w') \in \mathcal{I}(R_i)$, and
- if σ is finite with last letter σ_n, and $\rho(n) = w$, then w is terminal (i.e., there is no w' such that $w \prec w'$).

(Recall that $\mathcal{L}(w) \triangleq \{\mathtt{p} \mid \mathtt{p} \in \mathcal{I}(w)\}$ is the label of point w.) The fourth condition guarantees that generated words represent maximal paths in the model[5]. Define the *language generated by* $\mathcal{M}$ to be the set of all ω-words generated by $\mathcal{M}$. With these definitions, Kripke-models can be regarded as weakly fair transition systems for the alphabet $\Sigma = 2^{\mathcal{P}} \times \mathcal{R}$. (Recall that in a weakly fair transition system all states are recurring, and all terminal states are accepting.)

4.1. LEMMA. *For any Kripke-model $\mathcal{M} = (U, \mathcal{I}, w_0)$ there exists a weakly fair transition system $\mathcal{M}_{\mathcal{A}} = (S, \Delta, S_0)$, such that the language generated by $\mathcal{M}$ is equal to the language accepted by $\mathcal{M}_{\mathcal{A}}$.*

PROOF: To prove this lemma, there are several alternative constructions. One possibility is to define $S \triangleq U \cup \{\mathit{stop}\}$, where *stop* is a special accepting state for finite paths. Furthermore, $S_0 \triangleq \{w_0\}$, and $(w, (P, R), s) \in \Delta$ iff $w \in U$, $\mathcal{L}(w) = P$, and either $(w, s) \in \mathcal{I}(R)$ or w is terminal and $s = \mathit{stop}$. Then, $\mathcal{M}_{\mathcal{A}}$ accepts exactly the set of all natural models which are generated by $\mathcal{M}$. $\square$

Thus, models can be seen as automata. Likewise, formulas can be seen as automata: in the previous section we observed that for every **LTL** formula there exists an equivalent Büchi-automaton. Since this proof is constructive, it yields a method to obtain such an automaton. However, a much more concise way of constructing it is the tableau construction sketched in Section 7 below.

[5]Some texts omit this condition, with the consequence that all prefixes of a generated word are also generated. Other authors impose the even stronger condition that all generated words must be infinite; this implies that all points in a model should be nonterminal.

Let φ be an **LTL**-formula, and $\mathcal{M}$ be a Kripke-model with a single accessibility relation. Then φ is sequence-valid in $\mathcal{M}$ iff the language generated by $\mathcal{M}$ (i.e., the language accepted by the weakly fair transition system $\mathcal{M}_{\mathcal{A}}$ for $\mathcal{M}$) is a subset of the language accepted by the Büchi-automaton $\mathcal{M}_\varphi$ for φ. That is,

$$\mathcal{M} \models \varphi \quad \text{iff} \quad L(\mathcal{M}_{\mathcal{A}}) \subseteq L(\mathcal{M}_\varphi).$$

The latter condition is equivalent to $L(\mathcal{M}_{\mathcal{A}}) \cap \overline{L(\mathcal{M}_\varphi)} = \{\}$, or $L(\mathcal{M}_{\mathcal{A}} \times \mathcal{M}_{\neg\varphi}) = \{\}$. Here, $\mathcal{M}_1 \times \mathcal{M}_2$ denotes the product of ω-automata, where the product automaton $\mathcal{M}_1 \times \mathcal{M}_2$ accepts an infinite word σ iff each component automaton accepts σ. Formally, if $\mathcal{M}_i \triangleq (S_i, \Delta_i, S_{i,0}, S_{i,acc}, S_{i,rec})$ for $i = 1, 2$, then $\mathcal{M}_1 \times \mathcal{M}_2 \triangleq (S, \Delta, S_0, S_{acc}, S_{rec})$, where

- $S \triangleq S_1 \times S_2 \times \{1, 2\}$,
- $((s_1, s_2, i), a, (s_1', s_2', j)) \in \Delta$ iff $(s_1, a, s_1') \in \Delta_1$, $(s_2, a, s_2') \in \Delta_2$, and $i = j$, or $i = 1$ and $s_1 \in S_{1,rec}$ and $j = 2$, or $i = 2$ and $s_2 \in S_{2,rec}$ and $j = 1$.
- $S_0 \triangleq S_{1,0} \times S_{2,0} \times \{1\}$,
- $S_{acc} \triangleq S_{1,acc} \times S_{2,acc} \times \{0, 1\}$,
- $S_{rec} \triangleq S_{1,rec} \times S_{2,rec} \times \{2\}$,

Intuitively, the definition of S_{rec} enforces that in an infinite run of $\mathcal{M}_1 \times \mathcal{M}_2$ both a state from $S_{1,rec}$ and a state from $S_{2,rec}$ must be visited infinitely often. With this construction, model checking of **LTL** sequence-validity in finite models reduces to the nonemptyness problem of Büchi-automata: a feasible way to check whether $\mathcal{M} \models \varphi$ is to construct the Büchi-automata $\mathcal{M}_{\mathcal{A}}$ for the model and $\mathcal{M}_{\neg\varphi}$ for $\neg\varphi$, and to check whether the language of the product automaton $\mathcal{M}_{\mathcal{A}} \times \mathcal{M}_{\neg\varphi}$ is empty. This approach is implemented in the SPIN and COSPAN model checking tools [Holzmann 1991, Kurshan 1994].

If both system $\mathcal{M}$ and property φ are given as automata, then "specification" φ can be regarded as a "more abstract version" of the "implementation" $\mathcal{M}$. We write $\mathcal{M}_I \models \mathcal{M}_S$ if $L(\mathcal{M}_I) \subseteq L(\mathcal{M}_S)$, i.e., if (the language of) $\mathcal{M}_I$ is a subset of (the language of) $\mathcal{M}_S$. A *property* φ is defined to be just any ω-language $\varphi \subseteq \Sigma^\omega$, where $\Sigma = 2^{\mathcal{P}} \times \mathcal{R}$.

4.2. THEOREM. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be Büchi-automata. Then*

- *$\mathcal{M}_1 \models \mathcal{M}_2$ iff for all properties φ, if $\mathcal{M}_2 \models \varphi$ then $\mathcal{M}_1 \models \varphi$.*
- *$\mathcal{M}_1 \models \mathcal{M}_2$ iff for all ω-regular φ, if $\mathcal{M}_2 \models \varphi$ then $\mathcal{M}_1 \models \varphi$.*

PROOF: One direction is immediate by transitivity of the subset relation: if $L(\mathcal{M}_1) \subseteq L(\mathcal{M}_2)$ and $L(\mathcal{M}_2) \subseteq L(\varphi)$, then $L(\mathcal{M}_1) \subseteq L(\varphi)$. The other direction follows from instantiating φ with $L(\mathcal{M}_2)$ and, in the strong form, from the fact that the Büchi-automaton $\mathcal{M}_2$ defines a regular language. $\square$

This theorem can help to reduce the complexity of checking whether a model satisfies a formula. In order to prove $\mathcal{M}_1 \models \varphi$, it can be helpful to look for a "small" model $\mathcal{M}_2$ such that $\mathcal{M}_1 \models \mathcal{M}_2$ and $\mathcal{M}_2 \models \varphi$.

4.2. Safety and Liveness Properties

A similar characterization result as the above 4.2 holds for finite transition systems and a special class of ω-languages called *safety-properties*. For natural models $\mathcal{M}$ and $\mathcal{M}'$, let $\mathcal{M}^{[..i]}$ be the model consisting of the first i points of $\mathcal{M}$, and $\mathcal{M} \circ \mathcal{M}'$ be the concatenation of the two models $\mathcal{M}$ and $\mathcal{M}'$. (If $\mathcal{M}$ is infinite, then define $\mathcal{M} \circ \mathcal{M}' \triangleq \mathcal{M}$.)

- φ is a *safety property*, iff for every natural model $\mathcal{M}$,

$$\mathcal{M} \models \varphi \ \text{ if } \ \forall i \exists \mathcal{M}' : \ \mathcal{M}^{[..i]} \circ \mathcal{M}' \models \varphi$$

This definition is from [Alpern and Schneider 1985]. An ω-language φ is a safety property if for every model *not* satisfying φ there is a finite prefix $\mathcal{M}^{[..i]}$ which can not be completed by any continuation $\mathcal{M}'$ such that $\mathcal{M}^{[..i]} \circ \mathcal{M}' \models \varphi$. In other words, for every model dissatisfying φ something "bad" must have happened after some finite number of steps which cannot be remedied by any future good behavior. Hence, in Lamport's popular characterization, safety properties express that "something bad never happens" [Lamport 1983].

- φ is a *liveness property*, iff for every natural model $\mathcal{M}$,

$$\forall i \exists \mathcal{M}' : \ \mathcal{M}^{[..i]} \circ \mathcal{M}' \models \varphi$$

A liveness property φ, on the other hand, can never be refuted by observing only a finite prefix of some run. It holds, if and only if every finite sequence can be completed to a model satisfying φ, hence φ states that "something good eventually happens". Notice, however, that in contrast to the "bad thing" referred to above, the occurrence of the "good thing" does not have to be observable in any fixed time interval. Thus, liveness failures cannot be detected by testing.

Without proof we state some facts about safety and liveness from [Alpern and Schneider 1985]:

4.3. THEOREM. *(Properties of safety and liveness)*

- *Safety properties are closed under finite unions and arbitrary intersections.*
- *Liveness properties are closed under arbitrary unions, but not under intersections.*
- $\top$ *is the only property which is both a safety and a liveness property.*
- *For any property φ there exists a safety property φ_S and a liveness property φ_L such that $\varphi = (\varphi_S \cap \varphi_L)$.*

The last of these facts is known as the *decomposition theorem* and can be proved by topological arguments. The safety-part of a property φ is the topological closure of φ, that is, the least safety property containing φ. As an example, on natural models the **LTL**-formula $(\mathsf{p}\,\mathbf{U}^{+}\,\mathsf{q})$ is equivalent to $((\mathsf{p}\,\mathbf{W}^{+}\,\mathsf{q}) \wedge \mathbf{F}^{+}\,\mathsf{q})$, where the language defined by $(\mathsf{p}\,\mathbf{W}^{+}\,\mathsf{q})$ is a safety property and the language defined by $\mathbf{F}^{+}\,\mathsf{q}$ is a liveness property. Similarly, total correctness statements about programs can be decomposed into invariance (safety) and termination (liveness).

We now give a syntactical characterization of **LTL** safety properties.

4.4. THEOREM. *Every temporal formula built from literals with $\bot$, $\top$, $\wedge$, $\vee$ and $\mathbf{W}^+$ defines a safety property.*

PROOF: The proof is by induction on the structure of the formula. The only interesting case is $(\varphi\, \mathbf{W}^+\, \psi)$. Assume that any model $\mathcal{M}$ falsifying both φ and ψ has a finite prefix $\mathcal{M}^{[..i]}$ such that any extension of $\mathcal{M}^{[..i]}$ falsifies these formulas. If $\mathcal{M} \not\models (\varphi\, \mathbf{W}^+\, \psi)$, then there is a $w_j > w_0$ such that $w_j \models (\neg\varphi \wedge \neg\psi)$, and $w_k \models \neg\psi$ for $w_0 < w_k < w_j$. Therefore, in any model $\mathcal{M}^{[..j+i]} \circ \mathcal{M}'$, the formula $(\varphi\, \mathbf{W}^+\, \psi)$ must be invalid. ☐

An alternative characterization of safety in linear temporal logic is with past-operators. Any **LTL** formula $\mathbf{G}^*\psi$, where ψ is a past formula, defines a safety property. Moreover, any **LTL**-definable safety property can be defined by a formula of this form [Lichtenstein et al. 1985].

A binary relation $\Delta \subseteq U \times U$ is called *image finite*, if for any $x \in U$ the set $\{y \in U \mid (x,y) \in \Delta\}$ is finite. In particular, any finite relation is image finite. We call a transition system (S, Δ, S_0) *finitary*, if S_0 is finite and Δ is image finite. Of course, any finite transition system is finitary. Intuitively, finitary transition systems allow only "finite nondeterminism". The following statement extends Theorem 4.4 to finitary transition systems:

4.5. THEOREM. *Any finitary transition system defines a safety property.*

PROOF: Consider the language L of a finitary transition system. We have to show that for every sequence σ, if $\forall i \exists \sigma' : \ \sigma^{[..i]} \circ \sigma' \in L$ then $\sigma \in L$. In other words, assume that any finite prefix of σ can be extended to a string in L and show $\sigma \in L$. If σ is finite, then it is a finite prefix of itself; thus there exists some σ' such that $\sigma \circ \sigma' \in L$. Since every state of a transition system is accepting, it follows that $\sigma \in L$. If σ is infinite, consider the following computation tree: each node is marked by $(s, \sigma^{[..i]})$, where s is a state of the transition system and $\sigma^{[..i]}$ is a finite prefix of σ. The root is marked $(s, ())$, where s is any state. For any initial state $s_0 \in S_0$ of the transition system there is a child of the root in the computation tree which is marked (s_0, σ_0), where $\sigma_0 = \sigma^{[..0]}$ is the first letter of σ. Given a node marked $(s, \sigma^{[..i-1]})$ (where $i > 0$), for any s' such that $(s, \sigma_{i-1}, s') \in \Delta$ there is a child node in the tree marked $(s', (\sigma_0, ..., \sigma_i))$. Thus there exists a node marked $(s, \sigma^{[..i]})$ iff there is a path from some initial state to state s which is labelled by $(\sigma_0, ..., \sigma_{i-1})$. Since S_0 is finite and Δ is image finite, the computation tree is finitely branching. Since every prefix of σ can be extended to a string which is accepted by the transition system, the tree contains infinitely many nodes. Thus, by König's lemma from elementary set theory, it must contain an infinite branch. Therefore, there is a path in the transition system labelled by σ. Since all states in a transition system are recurring, it accepts σ. $\square$

$\mathcal{M}$:

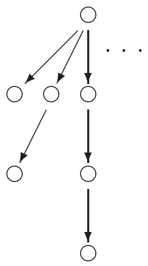


Figure 4: A non-finitary Kripke-model

Without the finitary restriction, Lemma 4.5 does not hold: consider the infinite transition system $\mathcal{M}$ of Figure 4. It shows a tree, such that for every natural number i a path of length i starts from the root. This transition system defines the set of all finite strings $(\mathbf{F}^{*}\, \mathbf{X} \perp)$, which is not a safety property. Similarly, the same language can be defined by an image finite transition system with infinitely many starting states. In particular, Lemma 4.5 implies that any finite transition system defines an ω-regular safety property. A weaker inverse statement also holds:

4.6. Lemma. *For every ω-regular safety property there is a finite transition system defining this property.*

Proof: Assume that a Büchi-automaton defining a certain safety property φ is given. We transform this automaton into a suitable normal form. First, any nonaccepting state s can either declared to be accepting or deleted, depending on whether an accepting state is reachable from s or not: since safety properties are prefix-closed languages, if there is an accepted path which passes through nonaccepting states, then there must be an equivalent path passing only through accepting states. Similarly, *nonaccepting SCCs* can be deleted: these are nontrivial strongly connected components in the automaton which do not contain a recurring state. Since φ is a safety property, for any accepted path ρ passing through states in a nonaccepting SCC there must be an equivalent path which avoids this SCC. Otherwise, assume that $\rho = \rho_1 \circ \rho_2$, where ρ_1 leads into the nonaccepting SCC. Consider the (nonaccepted) path $\rho_1 \circ \sigma^{\omega}$ which passes infinitely often through the nodes of this nonaccepting SCC. Any finite prefix $\rho_1 \circ \sigma^{n}$ of this path can be extended to the accepted path $\rho_1 \circ \sigma^{n} \circ \rho_2$; hence the whole path would have to be accepted. After the deletion of nonaccepting SCCs, each nontrivial SCC contains a recurring state. Therefore, the automaton accepts all finite and infinite paths through its state graph. Consider the transition system with the same state set and transition relation, where all states are accepting and recurring. The language of this transition system is the same as that of the (reduced) automaton. □

For **LTL** safety properties φ, a deterministic transition system $\mathcal{M}_\varphi$ corresponding to φ can be obtained directly by a tableau procedure; see section 7.

Given a finite Kripke model $\mathcal{M}$ and an ω-regular safety property φ, checking whether $\mathcal{M}$ sequence-validates φ is especially easy. Let $\mathcal{M}_\mathcal{A}$ be the weakly fair transition system corresponding to $\mathcal{M}$ according to Lemma 4.1, and let $\mathcal{M}_\varphi$ be a deterministic finite transition system defining the same language as φ. As above, $\mathcal{M} \models \varphi$ iff $L(\mathcal{M}_\mathcal{A}) \subseteq L(\mathcal{M}_\varphi)$. Language containment can be decided by executing $\mathcal{M}_\mathcal{A}$ (program) and $\mathcal{M}_\varphi$ (specification) in parallel and checking that for every step in $\mathcal{M}_\mathcal{A}$ the corresponding step in $\mathcal{M}_\varphi$ exists. This approach is also used in *specification-based testing*, where a number of *test runs* $\sigma \in L(\mathcal{M}_\mathcal{A})$ is checked whether they conform to the specification, that is, $\sigma \in L(\mathcal{M}_\varphi)$. The test runs are either determined by the system under test, or selected by the specification according to some coverage strategy.

Safety properties can be used to characterize language containment for finitary transition systems just as ω-regular properties for Büchi-automata (cf. Fact 4.2). For finitary transition systems, it is sufficient to check whether $\mathcal{M}_2 \models \varphi$ implies $\mathcal{M}_1 \models \varphi$ for all *safety* properties φ in order to establish $\mathcal{M}_1 \models \mathcal{M}_2$:

4.7. THEOREM. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be finitary transition systems. Then $\mathcal{M}_1 \models \mathcal{M}_2$ iff for all safety properties φ, if $\mathcal{M}_2 \models \varphi$ then $\mathcal{M}_1 \models \varphi$.*

PROOF: Assume that $\mathcal{M}_1 \models \mathcal{M}_2$, and that $\mathcal{M}_1 \not\models \varphi$. Then there exists a word σ accepted by $\mathcal{M}_1$ such that $\sigma \notin \varphi$. Since $L(\mathcal{M}_1) \subseteq L(\mathcal{M}_2)$, this counter model is also in the language of $\mathcal{M}_2$, hence $\mathcal{M}_2 \not\models \varphi$. For the other direction, since the set of all natural models generated from a finitary transition system is a safety property and by the fact that $\mathcal{M}_2 \models \mathcal{M}_2$ the assumption immediately reduces to $\mathcal{M}_1 \models \mathcal{M}_2$. □

4.3. Simulation Relations

The above characterization results concentrate on containment between the ω-languages generated by models and (linear time) formulas. However, there are two reasons to consider also weaker preorders between models than containment: firstly, for large nondeterministic transition systems language containment may not be easy to check. Secondly, sometimes it is desirable to formulate properties which depend on the *structure* of the system under consideration rather than on its *behavior*. Such properties may not be preserved even for systems generating the same language. For example, consider the two models $\mathcal{M}_1$ and $\mathcal{M}_2$ of Figure 5 over $\mathcal{P} = \{\}$ and $\mathcal{R} = \{a, b, c\}$.

Figure 5: Two sequence-equivalent but branching-inequivalent Kripke-models

Clearly, $L(\mathcal{M}_1) = L(\mathcal{M}_2)$, and therefore $\mathcal{M}_1 \models \mathcal{M}_2$. That is, if we observe sequences of transitions, then every possible behavior of $\mathcal{M}_1$ is also a possible behavior of $\mathcal{M}_2$. However, if we observe not only transitions which *are* taken, but also transitions which *could be* taken, then the behavior of $\mathcal{M}_1$ and $\mathcal{M}_2$ differs: if "possible continuations" are indicated by small light bulbs, then in the first system after performing a both the b and c lights will be lit, whereas in the second system only one of both is on. Formally, for every **LTL**-formula ψ it holds that ψ is sequence-valid in $\mathcal{M}_1$ iff ψ is sequence-valid in $\mathcal{M}_2$. For $\varphi \triangleq [a]([b]\perp \vee [c]\perp)$, it holds that $\mathcal{M}_2 \models \varphi$, but $\mathcal{M}_1 \not\models \varphi$.

Given two models $\mathcal{M}_1 = (U_1, \mathcal{I}_1, w_1)$ and $\mathcal{M}_2 = (U_2, \mathcal{I}_2, w_2)$, we say that $\mathcal{M}_1$ is a *submodel* of $\mathcal{M}_2$ (denoted by $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$), if $U_1 \subseteq U_2$, $\mathcal{I}_1 = \mathcal{I}_2 \downarrow U_1$ (the restriction of $\mathcal{I}_2$ to U_2), and $w_1 = w_2$. Intuitively, a submodel consists of some parts of the original model. In the proof of Lemma 4.6 we constructed a special submodel which preserves all execution sequences. Generally, all temporal properties are preserved when a model is replaced by the *generated submodel*, i.e., the submodel consisting of all points reachable from the current point. However, usually properties are not preserved when a model is replaced by an arbitrary submodel. Instead of simply omitting parts of a model, it is better to collapse several points into a single point.

For any two models $\mathcal{M}_1 = (U_1, \mathcal{I}_1, w_1)$ and $\mathcal{M}_2 = (U_2, \mathcal{I}_2, w_2)$, a relation $H \subseteq U_1 \times U_2$ is called a *simulation relation* between $\mathcal{M}_1$ and $\mathcal{M}_2$ if

- $(w_1, w_2) \in H$,
- For all $\mathtt{p} \in \mathcal{P}$, $u \in U_1$, and $v \in U_2$, if $(u, v) \in H$ then $u \in \mathcal{I}_1(\mathtt{p})$ iff $v \in \mathcal{I}_2(\mathtt{p})$.
- For all u and v such that $(u, v) \in H$ and all R and u' such that $(u, u') \in \mathcal{I}_1(R)$ there is a v' with the property that $(v, v') \in \mathcal{I}_2(R)$ and $(u', v') \in H$.

Figure 6 illustrates the third condition.

We say that $\mathcal{M}_1$ *is simulated by* $\mathcal{M}_2$, or $\mathcal{M}_2$ *simulates* $\mathcal{M}_1$ (denoted by $\mathcal{M}_1 \preceq \mathcal{M}_2$), if there exists a simulation relation H between $\mathcal{M}_1$ and $\mathcal{M}_2$. Simulation relates a model $\mathcal{M}_1$ to an *abstraction* $\mathcal{M}_2$ of the model $\mathcal{M}_1$. It guarantees that every behavior of the model is also a possible behavior of the abstraction. However, since a point in the abstract model usually represents a set of points in the original model, the abstraction might have behaviors that have no counterpart in the original model. Thus, the term "simulation" is used as in "the PC simulates a gameboy" or "this program simulates the development of bacteria cultures".

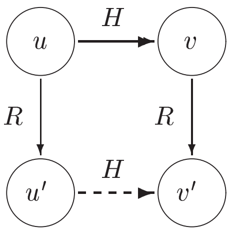


Figure 6: Simulation condition for u and v

4.8. FACT. $\underline{\rightarrow}$ is a preorder on the class of all models.

PROOF: The proof of reflexivity is immediate. For transitivity, note that the relational product of two simulation relations is again a simulation relation. $\square$

If $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$, then $\mathcal{M}_1 \underline{\rightarrow} \mathcal{M}_2$. Moreover, if $\mathcal{M}_1 \underline{\rightarrow} \mathcal{M}_2$, then $\mathcal{M}_1 \models \mathcal{M}_2$: if $\mathcal{M}_2$ can simulate $\mathcal{M}_1$, then for every maximal run σ generated by $\mathcal{M}_1$ there exists a corresponding $\sigma' \in \mathcal{M}_2$.

A model is called *deterministic*, if for every $w \in U$ and $R \in \mathcal{R}$ there is at most one $w' \in U$ such that $(w, w') \in \mathcal{I}(R)$. (This definition is somewhat weaker than the definition of deterministic automata on page 1668.) For deterministic $\mathcal{M}_2$ also the converse holds: $\mathcal{M}_1 \models \mathcal{M}_2$ iff $\mathcal{M}_1 \underline{\rightarrow} \mathcal{M}_2$. This is true because for any word there is at most one path through a deterministic transition system. Deterministic models and properties are an important special case. Whereas for many problems in nondeterministic transition systems an exponential search via backtracking is used, in the deterministic case the same problems can be solved with polynomial complexity.

4.9. LEMMA. *Let H be a simulation relation between $\mathcal{M}_1 = (U_1, \mathcal{I}_1, w_1)$ and $\mathcal{M}_2 = (U_2, \mathcal{I}_2, w_2)$, and $(w_1', w_2') \in H$. Then $(U_1, \mathcal{I}_1, w_1') \underline{\rightarrow} (U_2, \mathcal{I}_2, w_2')$.*

PROOF: The proof is immediate from the definition of simulation relations. $\square$

A *modal box formula* is a formula not involving any diamond operator. More precisely, literals (propositions and negated propositions) and $\bot, \top$ are modal box formulas, and if φ and ψ are modal box formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $[R]\varphi$ are modal box formulas. Similar to Lemmas 4.2 and 4.6, the following lemma relates simulations between models and preservation of modal box formulas:

4.10. LEMMA. *Let $\mathcal{M}_1 = (U_1, \mathcal{I}_1, w_1)$ and $\mathcal{M}_2 = (U_2, \mathcal{I}_2, w_2)$ be Kripke-models. $\mathcal{M}_1 \underline{\rightarrow} \mathcal{M}_2$ implies that for all modal box formulas φ, if $\mathcal{M}_2 \models \varphi$ then $\mathcal{M}_1 \models \varphi$.*

Proof: The proof is by induction on φ. The base cases $\bot, \top$ are trivial. For $\mathtt{p} \in \mathcal{P}$, the assumption $(w_1, w_2) \in H$ implies $w_1 \in \mathcal{I}_1(\mathtt{p})$ iff $w_2 \in \mathcal{I}_2(\mathtt{p})$. For boolean operators $\wedge$, $\vee$, the statement is an immediate consequence of the induction hypothesis. Finally, if $w_1 \not\models [R]\,\varphi$, then there is a $w_1' \in U_1$ such that $(w_1, w_1') \in \mathcal{I}_1(R)$ and $w_1' \not\models \varphi$. Since $\mathcal{M}_1 \stackrel{\rightarrow}{\rightarrow} \mathcal{M}_2$, there is a $w_2' \in U_2$ such that $(w_2, w_2') \in \mathcal{I}_2(R)$ and $(w_1', w_2') \in H$. Lemma 4.9 asserts that $(U_1, \mathcal{I}_1, w_1') \stackrel{\rightarrow}{\rightarrow} (U_2, \mathcal{I}_2, w_2')$. According to the induction hypothesis, $w_2' \not\models \varphi$. Therefore, $w_2 \not\models [R]\,\varphi$, which was to be proved. $\square$

This lemma makes it possible to check safety in the abstracted (small) model $\mathcal{M}_2$ rather than in the original (large) model $\mathcal{M}_1$: if $\mathcal{M}_1$ violates a modal box formula, then this violation will also occur in $\mathcal{M}_2$.

The above statement can be extended to more expressive logics. The logic **ACTL** [Long 1993, Clarke, Grumberg and Long 1994*a*, Clarke, Long and McMillan 1989, Josko 1993, Dams, Grumberg and Gerth 1994] is "**CTL** without **E** quantifier". That is, literals and $\top, \bot$ are **ACTL** formulas, and if φ and ψ are **ACTL** formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)$ and $\mathbf{A}(\varphi\,\mathbf{W}^{+}\,\psi)$ are **ACTL** formulas, where $\mathbf{A}(\varphi\,\mathbf{W}^{+}\,\psi) \triangleq \neg\mathbf{E}(\neg\psi\,\mathbf{U}^{+}\,\neg(\varphi \vee \psi))$.

4.11. Theorem. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be Kripke-models and φ be an* **ACTL** *formula. If $\mathcal{M}_1 \stackrel{\rightarrow}{\rightarrow} \mathcal{M}_2$ and $\mathcal{M}_2 \models \varphi$, then $\mathcal{M}_1 \models \varphi$.*

Proof: Intuitively, this theorem is true because formulas in **ACTL** describe properties that are valid in all paths of a model. They cannot express the existence of a specific path in the model. If $\mathcal{M}_1 \stackrel{\rightarrow}{\rightarrow} \mathcal{M}_2$, then every behavior of $\mathcal{M}_1$ is a behavior of $\mathcal{M}_2$. Thus every formula of **ACTL** that is valid in $\mathcal{M}_2$ must also be valid in $\mathcal{M}_1$.

Formally, the theorem is proved by induction on the structure of φ. Again, the only interesting cases are $\mathbf{A}\,\mathbf{U}^{+}$ and $\mathbf{A}\,\mathbf{W}^{+}$. We show the case of $\varphi \triangleq \mathbf{A}(\chi\,\mathbf{U}^{+}\,\psi)$. Note that $\neg\mathbf{A}(\chi\,\mathbf{U}^{+}\,\psi) \leftrightarrow (\mathbf{E}(\neg\psi\,\mathbf{U}^{+}\,\neg(\chi \vee \psi)) \vee \mathbf{E}\,\mathbf{G}^{+}\,\neg\psi)$ (cf. page 1655). Assume that $\mathcal{M}_1 \stackrel{\rightarrow}{\rightarrow} \mathcal{M}_2$ and $\mathcal{M}_1 \not\models \varphi$, and show that $\mathcal{M}_2 \not\models \varphi$. If $w_1 \not\models \mathbf{A}(\chi\,\mathbf{U}^{+}\,\psi)$, then in $\mathcal{M}_1$ there is either a finite sequence of nodes w_1^1, w_1^2, ..., w_1^n, such that $w_1^i \not\models \psi$ for $0 < i < n$, and $w_1^n \not\models (\chi \vee \psi)$, or a maximal path w_1^1, w_1^2, w_1^3, ..., such that $w_1^i \not\models \psi$ for all $i > 0$. Similar to the above, the induction hypothesis proves that a corresponding finite or infinite sequence w_2^1, w_2^2, ..., w_2^n or w_2^1, w_2^2, w_2^3, ..., exists, such that $w_2^i \not\models \psi$ for $0 < i < n$, and $w_2^n \not\models (\chi \vee \psi)$, or $w_2^i \not\models \psi$ for all $i > 0$. Thus $w_2 \not\models \mathbf{A}(\chi\,\mathbf{U}^{+}\,\psi)$. $\square$

In general the converse of the above lemma and theorem are not valid. Essentially, this is due to the same reason why Lemma 4.5 fails to hold for non-finitary transition system: consider the counterexample of Figure 7.

Both models have infinitely many branches from the root, one branch of length one, one branch of length two, one branch of length three, and so on. $\mathcal{M}_1$ has an additional branch of infinite length. These two models cannot be distinguished by any modal formula:

4.12. Lemma. *For any $\varphi \in$* **ML** *it holds that $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_2 \models \varphi$*

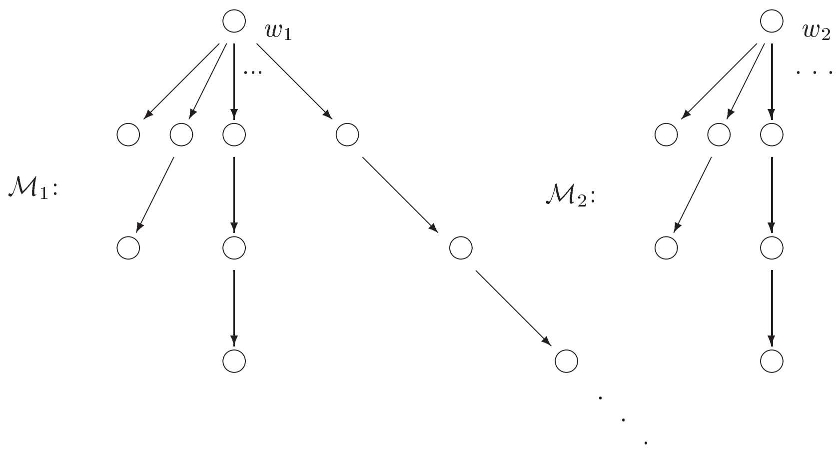


Figure 7: Two modally indistinguishable models

PROOF: The statement is proved by induction on φ. The crucial case is $\varphi = \langle R \rangle \psi$, $\mathcal{M}_1 \models \varphi$, and the successor w_1' of w_1 for which $w_1' \models \psi$ is on the additional infinite branch of $\mathcal{M}_1$. Choose any branch of $\mathcal{M}_2$ of length at least n, where n is the number of modal operators in φ. Denote the i-th point on the infinite branch of $\mathcal{M}_1$ and on the chosen branch of $\mathcal{M}_2$ by w_1^i and w_2^i, respectively (where $w_1^0 = w_1$ and $w_2^0 = w_2$). Then for all $i \leq n$ and all sub-formulas ξ_i of φ with at most $(n-i)$ modal operators it holds that $w_1^i \models \xi_i$ iff $w_2^i \models \xi_i$. This is proved by subinduction on $n-i$: if $n-i=0$, then it holds by definition of the models. If $n-i>0$ and $w_1^{i+1} \models \xi_{i+1}$ iff $w_2^{i+1} \models \xi_{i+1}$, then $w_1^i \models \langle R \rangle \xi_{i+1}$ iff $w_2^i \models \langle R \rangle \xi_{i+1}$. Especially, since φ has n modal operators, $w_1^0 \models \varphi$ iff $w_2^0 \models \varphi$. □

In particular, Lemma 4.12 implies that for every modal box formula φ, if $\mathcal{M}_2 \models \varphi$ then $\mathcal{M}_1 \models \varphi$. Yet, $\mathcal{M}_2$ does not simulate $\mathcal{M}_1$: assume a simulation relation H mapping the first node w of the infinite path of $\mathcal{M}_1$ to any node w' of any finite path in $\mathcal{M}_2$. Then H must map the successor of w to the successor of w', the successor of the successor of w to the successor of the successor of w', and so on. There are finitely many successors from w', but infinitely many successors from w. Thus, after a finite number of steps, there will be nodes $u \in \mathcal{M}_1$ and $v \in \mathcal{M}_2$ such that $(u, v) \in H$, and u has a successor in $\mathcal{M}_1$, but v has no successor in $\mathcal{M}_2$.

This is a somewhat contrived counterexample. In "many" cases, the converse will hold. Recall that a model is called *image finite*, if every point has only finitely many successors.

4.13. THEOREM. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be image finite Kripke-models. Then $\mathcal{M}_1 \rightrightarrows \mathcal{M}_2$ iff for all modal box formulas φ, if $\mathcal{M}_2 \models \varphi$ then $\mathcal{M}_1 \models \varphi$.*

PROOF: Assume that all modal box formulas holding in $\mathcal{M}_2 \triangleq (U_2, \mathcal{I}_2, w_2)$ are also valid for $\mathcal{M}_1 \triangleq (U_1, \mathcal{I}_1, w_1)$, and construct a simulation between $\mathcal{M}_1$ and $\mathcal{M}_2$. Define H by $(u,v) \in H$ iff for all modal box formulas φ, if $v \models \varphi$ then $u \models \varphi$. Then $(w_1, w_2) \in H$ by definition, and $(u,v) \in H$ implies $\mathcal{L}_1(u) = \mathcal{L}_2(v)$, since literals are modal box formulas. Assume $(u,v) \in H$ and $(u,u') \in \mathcal{I}_1(R)$. We have to show that there is a v' such that $(v,v') \in \mathcal{I}_2(R)$ and for all modal box formulas φ, if $u' \not\models \varphi$ then $v' \not\models \varphi$. Assume for contradiction that for each v' with $(v,v') \in \mathcal{I}_2(R)$ there is a $\varphi_{v'}$ such that $u' \not\models \varphi_{v'}$ and $v' \models \varphi_{v'}$. Since $\mathcal{M}_2$ is image finite, $\bigvee \varphi_{v'}$ exists and is a modal box formula. Moreover, for all such v', we have $v' \models \bigvee \varphi_{v'}$, which means $v \models [R] \bigvee \varphi_{v'}$. This implies $u \models [R] \bigvee \varphi_{v'}$ and therefore $u' \models \bigvee \varphi_{v'}$. This is a contradiction to the assumption that $u' \not\models \varphi_{v'}$ for all $\varphi_{v'}$. □

We already mentioned that the above theorems can be used to reduce the complexity of model checking. To prove that $\mathcal{M}_1 \models \varphi$, it can help to find an appropriate abstraction $\mathcal{M}_2$, and to prove $\mathcal{M}_1 \rightrightarrows \mathcal{M}_2$ and $\mathcal{M}_2 \models \varphi$. For more information, see [Bensalem, Bouajani, Loiseaux and Sifakis 1992].

Extremely efficient algorithms are known to check language inclusion for deterministic finite automata [Hopcroft and Ullman 1979]. These algorithms can be used to check the simulation preorder for deterministic models. For nondeterministic finite models $\mathcal{M}_1 = (U_1, \mathcal{I}_1, w_1)$ and $\mathcal{M}_2 = (U_2, \mathcal{I}_2, w_2)$, to check whether $\mathcal{M}_1 \rightrightarrows \mathcal{M}_2$ we define a sequence of relations H^0, H^1, ... on $U_1 \times U_2$ as follows:

- $(u,v) \in H^0$ iff for all $\mathtt{p} \in \mathcal{P}$ it holds that $u \in \mathcal{I}_1(\mathtt{p})$ iff $v \in \mathcal{I}_2(\mathtt{p})$
- $(u,v) \in H^{n+1}$ iff $(u,v) \in H^n$ and for all R and $u' \in U_1$ such that $(u,u') \in \mathcal{I}_1(R)$ there is a v' with the property that $(v,v') \in \mathcal{I}_2(R)$ and $(u',v') \in H^n$.

The intersection H^* of all H^n is the largest simulation relation between $\mathcal{M}_1$ and $\mathcal{M}_2$. That is, $\mathcal{M}_1 \rightrightarrows \mathcal{M}_2$ iff $(w_1, w_2) \in H^*$. Algorithmically, if $H^n = H^{n-1}$, then $H^* \triangleq H^n$ and the construction terminates. In other words, we construct the greatest fixed point of the one-step simulation relation. Since the structures are finite, there are only finitely many different H^n. Thus, termination is guaranteed. In Figure 8, $R(u)$ denotes the set $\{u' \mid (u,u') \in \mathcal{I}(R)\}$, and $|_1$ is the first component of a tuple. In the next section, a more elaborate implementation of a similar algorithm for symmetric simulation relations is given, which is based on partition refinement.

5. Equivalence reductions

In this section, we consider symmetric preorders, i.e., equivalences, and equivalence transformations between models. There are various possibilities for defining equivalences on models. For any preorder $\preceq$ from the preceding section, an equivalence can be defined by $\mathcal{M}_1 \simeq \mathcal{M}_2$ iff $\mathcal{M}_1 \preceq \mathcal{M}_2$ and $\mathcal{M}_2 \preceq \mathcal{M}_1$. In this way, the equivalence induced by the submodel ordering $\sqsubseteq$ is isomorphism. For $\mathcal{M}_1 \models \mathcal{M}_2$, the

procedure `Sim_check` (Model $(U_1, \mathcal{I}_1, w_1)$, Model $(U_2, \mathcal{I}_2, w_2)$) =
  $H^{new} := \{(u, v) \mid u \in U_1,\ v \in U_2,\ \mathcal{L}_1(u) = \mathcal{L}_2(v)\}$
  repeat
    $H^{old} := H^{new}$; $H^{new} := \{\}$
    for all $(u, v) \in H^{old}$ **do**
      $add := \top$; **for all** $R \in \mathcal{R}$ **do**
        if not $R(u) \subseteq \big((R(u) \times R(v)) \cap H^{old}\big)|_1$ **then** $add := \bot$
      if add **then** $H^{new} := H^{new} \cup \{(u, v)\}$
  until $H^{new} = H^{old}$;
  if $(w_1, w_2) \in H^{new}$
  then `print`("$(U_1, \mathcal{I}_1, w_1)$ is simulated by $(U_2, \mathcal{I}_2, w_2)$")
  else `print`("There is no simulation between $(U_1, \mathcal{I}_1, w_1)$ and $(U_2, \mathcal{I}_2, w_2)$");

Figure 8: Algorithm for simulation checking

symmetric version is equality of the generated languages. Other model equivalences are introduced by equivalence with respect to logical formulas, and by symmetric simulations.

5.1. *Bisimulations (p-morphisms)*

A classical notion from modal logic is *p-morphism* [Segerberg 1968], [Segerberg 1971, p37] or *bisimulation* [Milner 1980, Park 1981]. A bisimulation is a relation $\leftrightarroweq$ between the universes of two Kripke-models $(U_1, \mathcal{I}_1, w_1)$ and $(U_2, \mathcal{I}_2, w_2)$ such that

- $w_1 \leftrightarroweq w_2$,
- If $u \leftrightarroweq v$, then $u \in \mathcal{I}_1(\mathtt{p})$ iff $v \in \mathcal{I}_2(\mathtt{p})$
- If $u \leftrightarroweq v$ and $(u, u') \in \mathcal{I}_1(R)$, then there exists v' such that $(v, v') \in \mathcal{I}_2(R)$ and $u' \leftrightarroweq v'$.
- If $u \leftrightarroweq v$ and $(v, v') \in \mathcal{I}_2(R)$, then there exists u' such that $(u, u') \in \mathcal{I}_1(R)$ and $u' \leftrightarroweq v'$.

Two Kripke-models $\mathcal{M}_1$ and $\mathcal{M}_2$ are *bisimilar* (denoted by $\mathcal{M}_1 \leftrightarroweq \mathcal{M}_2$), if there exists a bisimulation between them. Figure 9 shows some examples of bisimilar models.

This example demonstrates the following statements:

5.1. FACT.
- Each model is bisimilar to one where duplicate states (which have the same input and output) are removed,
- Each model is bisimilar to its unfolding, and

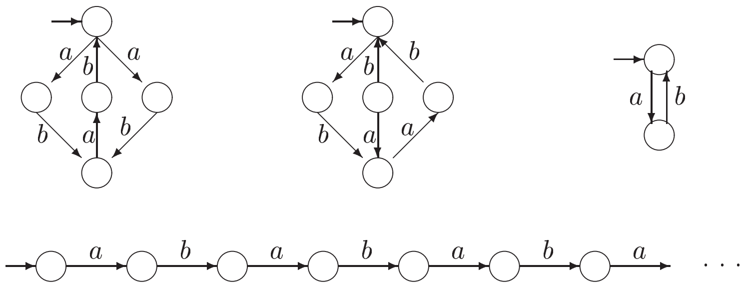


Figure 9: Bisimilar models

- Each model is bisimilar to its reachable part.

If $\mathcal{M}_1 \;\underline{\leftrightarrow}\; \mathcal{M}_2$, then $\mathcal{M}_1 \;\underline{\rightarrow}\; \mathcal{M}_2$ and $\mathcal{M}_2 \;\underline{\rightarrow}\; \mathcal{M}_1$; the other direction of this statement is not necessarily true. For example, each of the models in Figure 10 simulates the other one, but they are not bisimilar.

Figure 10: Not-bisimilar models

Another important equivalence relation between models is that of being indistinguishable by formulas of a certain logic. We say that the models $\mathcal{M}_1$ and $\mathcal{M}_2$ are *equivalent with respect to the logic* $\mathbf{L}$ ($\mathcal{M}_1 \equiv_{\mathbf{L}} \mathcal{M}_2$) if for all well formed formulas of $\mathbf{L}$ it holds that $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_2 \models \varphi$. The relation $\equiv_{FOL}$ is called *elementary equivalence*. Bisimulation relations are precisely those equivalences which preserve all modal formulas:

5.2. Lemma. *Bisimilar models are modally equivalent: if* $\mathcal{M}_1 \;\underline{\leftrightarrow}\; \mathcal{M}_2$*, then* $\mathcal{M}_1 \equiv_{\mathbf{ML}} \mathcal{M}_2$

The proof is by induction on the structure of φ, analogous to the proof of Lemma 4.10.

Hence, it is "safe" to substitute a model by a bisimilar one in a structured software development process: all multimodal formulas which are valid for the original model will remain valid for the substituted model. The converse of this lemma again requires image finiteness:

5.3. THEOREM (Segerberg71). *Image finite models are modally equivalent iff they are bisimilar: if* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *are image finite, then* $\mathcal{M}_1 \leftrightarrows \mathcal{M}_2$ *iff* $\mathcal{M}_1 \equiv_{\mathbf{ML}} \mathcal{M}_2$

Again, the proof is similar to the proof of Theorem 4.13 in the previous section. The only difference is that bisimulation is a symmetric relation.

In general, this theorem does not hold for more expressive logics. For *finite* Kripke-models, however, it can be lifted even to logics like positive $\mu\mathbf{TL}$. Given any formula φ which is positive in q, and a natural number n, we define $\nu^0 q\ \varphi \triangleq \top$, and $\nu^{n+1} q\ \varphi \triangleq \varphi\{q := \nu^n q\ \varphi\}$. That is, $\nu^n q\ \varphi \triangleq \varphi\{q := \varphi\}\{q := \varphi\} \cdots \{q := \top\}$.

5.4. LEMMA. *Let* $\mathcal{M} \triangleq (U, \mathcal{I}, w)$ *be a finite model, where* $|U| = n$, *and let* φ *be a monotonic* $\mu\mathbf{TL}$ *formula. Then* $\mathcal{M} \models \nu q\ \varphi$ *iff* $\mathcal{M} \models \nu^n q\ \varphi$

PROOF: One direction of this lemma follows from the fact that $\nu q\ \varphi$ denotes a fixed point, i.e., $(\nu q\ \varphi \to \varphi\{q := \nu q\ \varphi\})$. Since φ is monotonic, this implies $(\varphi\{q := \nu q\ \varphi\} \to \varphi\{q := \varphi\{q := \nu q\ \varphi\}\})$. By chain reasoning, $(\nu q\ \varphi \to \varphi\{q := \varphi\{q := \nu q\ \varphi\}\})$. By induction, $(\nu q\ \varphi \to \varphi\{q := \varphi\}\{q := \varphi\}...\{q := \nu q\ \varphi\})$. Again, since φ is monotonic in q, it holds that $(\varphi\{q := \nu q\ \varphi\} \to \varphi\{q := \top\})$, thus $(\nu q\ \varphi \to \nu^n q\ \varphi)$ is valid.

For the other direction, let $\mathcal{F} \triangleq (U, \mathcal{I})$ be the frame on which $\mathcal{M}$ is based. Consider the sequence $((\nu^n q\ \varphi)^{\mathcal{F}})_{n \geq 0}$ of sets of points. Clearly, $(\nu^0 q\ \varphi)^{\mathcal{F}} = \top^{\mathcal{F}} = U \supseteq (\nu^1 q\ \varphi)^{\mathcal{F}}$. Since $\varphi_q^{\mathcal{F}}$ is monotonic (cf. Fact 3.8), $(\nu^1 q\ \varphi)^{\mathcal{F}} = \varphi_q^{\mathcal{F}}((\nu^0 q\ \varphi)^{\mathcal{F}}) \supseteq \varphi_q^{\mathcal{F}}((\nu^1 q\ \varphi)^{\mathcal{F}}) = (\nu^2 q\ \varphi)^{\mathcal{F}}$. Continuing this argument, we conclude that $((\nu^n q\ \varphi)^{\mathcal{F}})_{n \geq 0}$ is a descending chain of sets. There are two possibilities: either there exists an $i < |U|$ such that $(\nu^i q\ \varphi)^{\mathcal{F}} = (\nu^{i+1} q\ \varphi)^{\mathcal{F}}$, hence $(\nu^i q\ \varphi)^{\mathcal{F}} = (\nu^n q\ \varphi)^{\mathcal{F}}$, or $(\nu^n q\ \varphi)^{\mathcal{F}} = \{\ \}$. In either case, the sequence stabilizes after at most $|U|$ steps: $(\nu^n q\ \varphi)^{\mathcal{F}} = (\nu^{n+1} q\ \varphi)^{\mathcal{F}}$. As a consequence, $(\nu^n q\ \varphi \to \nu^{n+1} q\ \varphi)$ is universally valid in $\mathcal{F}$.

Now assume that $\mathcal{M} \not\models \nu q\ \varphi$, and show that $\mathcal{M} \not\models \nu^n q\ \varphi$. According to the definition on page 1660, $(U, \mathcal{I}, w) \not\models \nu q\ \varphi$ means that for all $Q \subseteq U$ such that $w \in Q$ there exists a $v \in Q$ such that $(U, \mathcal{I}', v) \not\models \varphi$, where $\mathcal{I}'(q) = Q$. (*) Let $Q = (\nu^n q\ \varphi)^{\mathcal{F}}$. If $w \notin Q$, then $(U, \mathcal{I}, w) \not\models \nu^n q\ \varphi$ and we are done. If $w \in Q$, then by (*) for some v it holds that $(U, \mathcal{I}, v) \models \nu^n q\ \varphi$, and $(U, \mathcal{I}', v) \not\models \varphi$, where $\mathcal{I}'(q) = (\nu^n q\ \varphi)^{\mathcal{F}}$. In other words, $(U, \mathcal{I}', v) \not\models \varphi\{q := \nu^n q\ \varphi\}$, which means that $(U, \mathcal{I}, v) \not\models \nu^{n+1}\varphi$. Since $(U, \mathcal{I}, v) \models (\nu^n q\ \varphi \to \nu^{n+1} q\ \varphi)$, this is a contradiction. $\Box$

This lemma is important for model checking of $\mu\mathbf{TL}$ on finite Kripke models. Moreover, it allows to prove the following result.

5.5. THEOREM. *Finite models are monotonic* $\mu\mathbf{TL}$*-equivalent iff they are bisimilar: if* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *are finite, then* $\mathcal{M}_1 \leftrightarrows \mathcal{M}_2$ *iff* $\mathcal{M}_1 \equiv_{\mu\mathbf{TL}} \mathcal{M}_2$

PROOF: Trivially, finite models are also image finite. Any two models which are equivalent with respect to monotonic $\mu\mathbf{TL}$ are modally equivalent, since modal logic is a sublanguage of $\mu\mathbf{TL}$. Hence as an immediate consequence of Theorem 5.3, any two finite models which are $\mu\mathbf{TL}$ equivalent are bisimilar.

For the other direction, assume that $\mathcal{M}_1 \models \varphi$ and $\mathcal{M}_2 \not\models \varphi$, where φ is a monotonic μ**TL**-formula. Let $n \triangleq \max(|U_1|, |U_2|)$, and φ^n be φ where every subformula $\nu q\ \psi$ is replaced by $\nu^n q\ \psi$. As a consequence of Lemma 5.4, $\mathcal{M}_i \models \varphi$ iff $\mathcal{M}_i \models \varphi^n$ for $i = 1, 2$. Therefore, $\mathcal{M}_1 \models \varphi^n$ and $\mathcal{M}_2 \not\models \varphi^n$. Since φ^n is a multimodal formula, $\mathcal{M}_1$ and $\mathcal{M}_2$ are modally inequivalent. Theorem 5.3 implies that $\mathcal{M}_1$ and $\mathcal{M}_2$ are not bisimilar. □

5.6. COROLLARY. *Any two finite Kripke-models which can be distinguished by a monotonic μ**TL**-formula can also be distinguished by a multimodal formula: if $\mathcal{M}_1$ and $\mathcal{M}_2$ are finite, then $\mathcal{M}_1 \equiv_{\mu\mathbf{TL}} \mathcal{M}_2$ iff $\mathcal{M}_1 \equiv_{\mathbf{ML}} \mathcal{M}_2$*

[Browne, Clarke and Grumberg 1988] proved that if two finite models can be distinguished by a formula of the logic **CTL***, then they can be distinguished by a **CTL** formula. Every **CTL*** formula has a positive μ**TL** equivalent [Dam 1994] (on tree models, **CTL*** can be translated into monadic second order logic, which is of the same expressiveness as μ**TL**). Therefore this result can be obtained as a consequence of the above.

5.2. Distinguishing Power and Ehrenfeucht-Fraïssé Games

The previous theorems showed that logics with different expressiveness can have the same distinguishing capabilities. We wish to formalize these notions. A logic **L2** is said to be *at least as expressive as* **L1** (or **L1** is *at most as expressive as* **L2**) iff for any formula $\varphi_1 \in$ **L1** there exists a formula $\varphi_2 \in$ **L2** such that for all models $\mathcal{M}$ we have $\mathcal{M} \models \varphi_1$ iff $\mathcal{M} \models \varphi_2$. **L1** and **L2** *have the same expressive power* if **L1** is at least as expressive as **L2** and **L2** is at least as expressive as **L1**. In other words, two logics have the same expressive power iff for any formula of one logic there is an equivalent formula from the other logic. For example, Theorem 2.4 states that on natural models, **FOL** and **LTL** have the same expressive power.

Logic **L2** is *at least as distinguishing as* **L1** (or **L1** is *at most as distinguishing as* **L2**) if any two models which are inequivalent with respect to **L1** are also inequivalent with respect to **L2**. That is, **L2** at least as distinguishing as **L1** iff $\mathcal{M}_1 \equiv_{\mathbf{L2}} \mathcal{M}_2$ implies $\mathcal{M}_1 \equiv_{\mathbf{L1}} \mathcal{M}_2$. **L1** and **L2** *have the same distinguishing power* if **L1** is at most as distinguishing as **L2** and **L2** is at most as distinguishing as **L1**. In other words, **L1** and **L2** have the same distinguishing power iff for all models $\mathcal{M}_1$ and $\mathcal{M}_2$ it holds that $\mathcal{M}_1 \equiv_{\mathbf{L1}} \mathcal{M}_2$ iff $\mathcal{M}_1 \equiv_{\mathbf{L2}} \mathcal{M}_2$.

Expressiveness is a finer equivalence relation on the class of all logics than distinguishability:

5.7. FACT. If **L1** is at most as expressive as **L2**, then it is at most as distinguishing. If **L1** and **L2** have the same expressive power, then they have the same distinguishing power (but not vice versa).

PROOF: Assume that for any formula $\varphi_1 \in$ **L1** there exists an equivalent formula $\varphi_2 \in$ **L2**. Assume further two models $\mathcal{M}_1$ and $\mathcal{M}_2$ which are inequivalent with respect to **L1**, that is, for some $\varphi_1 \in \mathbf{L1}$ we have $\mathcal{M}_1 \models \varphi_1$ and $\mathcal{M}_2 \not\models \varphi_1$ or vice versa. According to the first assumption there exists $\varphi_2 \in \mathbf{L2}$ equivalent to φ_1. Therefore $\mathcal{M}_1 \models \varphi_2$ and $\mathcal{M}_2 \not\models \varphi_2$ or vice versa, which means that $\mathcal{M}_1$ and $\mathcal{M}_2$ are inequivalent with respect to **L2**. The second statement follows by symmetry. As example of logics with equal distinguishing power but different expressive power, consider multimodal logic and positive μ**TL**. □

For any formula φ, we say that φ is *preserved under bisimulations*, if for all models $\mathcal{M}_1 \leftrightarrows \mathcal{M}_2$ it holds that $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_2 \models \varphi$. A logic **L** is *bisimulation invariant*, if all well-formed formulas of **L** are preserved under bisimulations. Lemma 5.2 shows that multimodal logic is bisimulation invariant. In other words, if a property can be defined by a multimodal formula, then it is preserved under bisimulations. The same holds for more expressive logics like monotonic μ**TL**:

5.8. Lemma. μ**TL** *is bisimulation invariant: if $\mathcal{M}_1 \leftrightarrows \mathcal{M}_2$, then for any positive* μ**TL** *formula φ it holds that $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_2 \models \varphi$.*

In his thesis, van Benthem investigated the reverse direction, and gave a connection between bisimulations, first order and modal expressiveness (see [van Benthem 1983]). He showed that for first order formulas, bisimulation invariance implies multimodal definability:

5.9. Theorem (Expressive completeness of **ML**). *For any first order formula φ (with one free variable) which is preserved under bisimulations there exists an equivalent multimodal formula.*

Thus, exactly those first order formulas which are preserved under bisimulations can be translated into modal logic. [Janin and Walukiewicz 1996] extended this theorem for second order formulas and μ**TL**, which is a converse to Lemma 5.8:

5.10. Theorem (Expressive completeness of μ**TL**). *Let φ be any* **MSOL** *property. Then φ is preserved under bisimulations iff φ is definable by a positive* μ**TL** *formula.*

In particular, this result implies that every logic which is bisimulation invariant and has a semantical translation into **MSOL** can be also translated into mTL. As a corollary, many propositional logics of programs (CTL*, PDL, ...) which have been suggested can be translated into the μ-calculus.

Segerberg's theorem 5.3 relates modal equivalence to bisimilarity. Bisimilarity can also be defined in terms of a so-called *Ehrenfeucht-Fraïssé game* [Fraïssé 1954, Ehrenfeucht 1961]: there are two players, Ann and Bob. They play on a board on which two Kripke-models are drawn. Ann's goal is to show that these models are not bisimilar, whereas Bob's goal is to show that they are bisimilar. (So, this is not really a fair game, since the outcome is predetermined by the shape of the board.)

Each player has an unlimited amount of pebbles which are numbered consecutively: $a_0, a_1, a_2, \ldots$ and $b_0, b_1, b_2, \ldots$. To start the game, each player places his first pebble a_0, b_0 on the current point of one of the models. If the current points have a different label, Bob has lost immediately.

Thus, round 0 consists of placing a_0 and b_0 on the board. Similarly, round j consists of the placement of a_j and b_j: Ann chooses any point w_0 on one of the models on which some pebble (say, a_i or b_i for $i < j$) had been placed previously, and puts her next pebble a_j on some point w_1 which is an R-successor of w_0. Bob then locates the i^{th} pebble (that is, b_i or a_i, respectively) on the other model, say in point w'_0. He looks for a point w'_1 such that $w'_0 R w'_1$, and w_1 and $w_{1'}$ have the same label. If he can't find such a point he has lost and the game ends; otherwise he chooses any such point and puts his next pebble b_j on it.

If the game continues forever, then Bob has won. Ann *can force a win within n rounds*, if she can place her pebble in such a way that Bob immediately loses the game, or if she can choose a point such that for each possible answer of Bob she can force a win within $n-1$ rounds. Ann has a *winning strategy* if there is some n such that she can force a win within n rounds. Bob has winning strategy iff Ann does not have one; i.e., if in each round and for each possible move of Ann there is a response by Bob to continue the game.

Ehrenfeucht-Fraïssé games are a convenient way to imagine bisimulations.

5.11. Theorem. *Ann has a winning strategy in this game iff the two models are not bisimilar; i.e., Bob has a winning strategy iff they are bisimilar.*

Proof: From Bob's winning strategy, it is easy to construct a bisimulation between the two models: $w_i \leftrightarroweq w'_i$ iff Bob would have chosen w_i or w'_i as a reply to Ann's choosing w'_i or w_i, respectively. For the other direction, every bisimulation determines a winning strategy for Bob: he just replies by choosing any point which is related to the point chosen by Ann via the bisimulation relation. $\square$

It is easy to modify the rules of the game such that it captures the equivalence of two models with respect to other logical languages. For example, in a game for **MSOL** we allow both Ann and Bob in any move to place an arbitrary *set* of pebbles on one of the models on the board. Then the two models can be distinguished by a monadic second order formula iff Ann has a winning strategy.

5.3. Auto-bisimulations and the Paige/Tarjan Algorithm

In this subsection we show how to minimize a given Kripke-model with respect to bisimulation equivalence. Note that our definitions did not exclude bisimulations from a model to itself (*auto-bisimulations*); i.e., some points in a model can be related by a bisimulation to other points in the *same* model.

5.12. Lemma. *The union of any number of auto-bisimulations on a model is again an auto-bisimulation.*

Proof: This follows directly from the definition of bisimulation relations. $\square$

Thus, for any model, there exists a largest auto-bisimulation, namely, the union of all auto-bisimulations of this model. Additionally, the reflexive transitive symmetric closure of any auto-bisimulation is again an auto-bisimulation. Hence, for any auto-bisimulation $\leftrightarroweq$ there is a largest equivalence relation $\equiv$ containing it ($\leftrightarroweq \subseteq \equiv$) which is again an auto-bisimulation. And, the largest auto-bisimulation must be an equivalence relation on the set of points of a model.

Given any model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$, and any equivalence relation $\equiv$ on U. Define the *quotient of* $\mathcal{M}$ *with respect to* $\equiv$ to be the model $\mathcal{M}^{\equiv} \triangleq (U^{\equiv}, \mathcal{I}^{\equiv}, w_0^{\equiv})$, where $U^{\equiv}$ is the set of equivalence classes of U with respect to $\equiv$, $w_0^{\equiv}$ is the equivalence class of w_0, $w^{\equiv} \in \mathcal{I}^{\equiv}(\mathtt{p})$ if there is some $w \in w^{\equiv}$ such that $w \in \mathcal{I}(\mathtt{p})$, and $(w_1^{\equiv}, w_2^{\equiv}) \in \mathcal{I}^{\equiv}(R)$ if there are $w_1 \in w_1^{\equiv}$ and $w_2 \in w_2^{\equiv}$ such that $(w_1, w_2) \in \mathcal{I}(R)$.

5.13. LEMMA. *If the equivalence relation $\equiv$ is an auto-bisimulation, then $\mathcal{M} \leftrightarroweq \mathcal{M}^{\equiv}$.*

PROOF: Define $u \leftrightarroweq v^{\equiv}$ iff $u \equiv v$. That is, each point in the original model is mapped to its equivalence class in the quotient model. We have to show that for this relation the four conditions defining a bisimulation (cf. page 1682) hold. For the initial point, $w_0 \leftrightarroweq w_0^{\equiv}$ holds because $w_0 \equiv w_0$. Since $\equiv$ is a bisimulation, $u \equiv v$ implies that $\mathcal{L}(u) = \mathcal{L}(v)$. Thus if $u \leftrightarroweq v^{\equiv}$ then $u \in \mathcal{I}(\mathtt{p})$ iff $v^{\equiv} \in \mathcal{I}^{\equiv}(\mathtt{p})$. Furthermore, if $(u_1, u_2) \in \mathcal{I}(R)$ and $u_1 \leftrightarroweq v_1^{\equiv}$, then by definition $(u_1^{\equiv}, u_2^{\equiv}) \in \mathcal{I}^{\equiv}(R)$ and $u_1 \equiv v_1$. Therefore, $u_1^{\equiv} = v_1^{\equiv}$, i.e., $(v_1^{\equiv}, u_2^{\equiv}) \in \mathcal{I}^{\equiv}(R)$. For the last condition, assume that $(v_1^{\equiv}, v_2^{\equiv}) \in \mathcal{I}^{\equiv}(R)$ and $v_1^{\equiv} \leftrightarroweq u_1$. Then there exist w_1 and w_2 such that $w_1 \equiv v_1$, $w_2 \equiv v_2$ and $(w_1, w_2) \in \mathcal{I}(R)$. From $v_1^{\equiv} \leftrightarroweq u_1$ we infer $u_1 \equiv v_1$ and thus $u_1 \equiv w_1$. Since $\equiv$ is a bisimulation, there exists a $u_2 \equiv w_2$ such that $(u_1, u_2) \in \mathcal{I}(R)$. From $u_2 \equiv w_2$ and $w_2 \equiv v_2$ we conclude that $u_2 \equiv v_2$, i.e., $u_2 \leftrightarroweq v_2^{\equiv}$. □

The quotient of a model with respect to its largest auto-bisimulation can be regarded as a minimal representation of this model. In finite models, this minimal representation can be constructed very efficiently.

For any set of points $P \subseteq U$, let $\langle R \rangle P \triangleq \{w \mid \exists w' \in P, (w, w') \in \mathcal{I}(R)\}$. Given any partition of U into equivalence classes, call a component $w^{\equiv}$ *uniform*, if for all $\mathtt{p} \in \mathcal{P}$ it holds that $w^{\equiv} \subseteq \mathcal{I}(\mathtt{p})$ or $w^{\equiv} \cap \mathcal{I}(\mathtt{p}) = \{\}$. That is, $w^{\equiv}$ is uniform if $\mathcal{L}(w_1) = \mathcal{L}(w_2)$ for all $w_1, w_2 \in w^{\equiv}$. A component $w^{\equiv}$ is called *stable with respect to* P, if for all R either $w^{\equiv} \subseteq \langle R \rangle P$ or $w^{\equiv} \cap \langle R \rangle P = \{\}$. The partition is called *stable*, if all components are uniform and stable with respect to all components.

5.14. THEOREM. *The coarsest stable partition is the largest auto-bisimulation.*

PROOF: First, we show that any stable partition is an auto-bisimulation. Trivially, $w_0 \equiv w_0$. Since $u^{\equiv}$ is uniform, $u \equiv v$ implies $\mathcal{L}(u) = \mathcal{L}(v)$. If $(u, u') \in \mathcal{I}(R)$, then $u^{\equiv} \subseteq \langle R \rangle u'^{\equiv}$, because $u^{\equiv}$ is stable with respect to $u'^{\equiv}$. In other words, $u^{\equiv} \subseteq \{w \mid \exists w' \equiv u',\ (w, w') \in \mathcal{I}(R)\}$. Therefore, if $u \equiv v$, then there is a $v' \equiv u'$ such that $(v, v') \in \mathcal{I}(R)$. The symmetric condition is proved symmetrically. Vice versa, every auto-bisimulation defines a stable partition: to show that $u^{\equiv}$ is stable with respect to $v^{\equiv}$, assume that $u_1 \equiv u_2 \in u^{\equiv}$. Since $\equiv$ is a bisimulation, for every $(u_1, u_1') \in \mathcal{I}(R)$

and $u_1' \in v^{\equiv}$ there must be a $u_2' \equiv u_1' \in v^{\equiv}$ such that $(u_2, u_2') \in \mathcal{I}(R)$. Therefore, $u^{\equiv} \subseteq \langle R\rangle\, v^{\equiv}$ or $u^{\equiv} \cap \langle R\rangle\, v^{\equiv} = \{\}$. If $\equiv$ is the coarsest stable partition, then for any auto-bisimulation $\leftrightarrows$ it holds that $\leftrightarrows \;\subseteq\; \equiv$. Assuming for contradiction that u, v and $\leftrightarrows$ exist such that $u \leftrightarrows v$ and not $u \equiv v$, according to Lemma 5.12 the union of $\leftrightarrows$ and $\equiv$ would be a stable partition coarser than $\equiv$. □

The following algorithm can be used to construct the coarsest stable partition:

— Start with the trivial partition consisting of only one component

— Repeat

- Choose a component $w_0^{\equiv}$ and a proposition $\mathtt{p} \in \mathcal{P}$;
- Split $w_0^{\equiv}$ into $w_0^{\equiv} \cap \mathcal{I}(\mathtt{p})$ and $w_0^{\equiv} \setminus \mathcal{I}(\mathtt{p})$

or

- Choose components $w_0^{\equiv}$ and $w_1^{\equiv}$, and a relation $R \in \mathcal{R}$;
- Split $w_0^{\equiv}$ into $w_0^{\equiv} \cap \langle R\rangle\, w_1^{\equiv}$ and $w_0^{\equiv} \setminus \langle R\rangle\, w_1^{\equiv}$

until no new components can be obtained that way

The Paige-Tarjan algorithm [Paige and Tarjan 1987] given in Figure 11 is a sophisticated implementation of this idea; it maintains two partitions: a coarser one, C, and a finer one, F. All components in F are stable with respect to any component in C. The nondeterministic choice in the above *repeat*-loop is replaced by a systematic split of the finer partition with respect to all components of the coarser partition. Initially, C is the trivial partition and F is the split of C w.r.t. all $\mathtt{p} \in \mathcal{P}$ and $R \in \mathcal{R}$. Then, a $w^{\equiv} \in C$ is split into $w_1^{\equiv} \in F$ and $w_2^{\equiv} \triangleq w^{\equiv} \setminus w_1^{\equiv}$. Any $w_0^{\equiv} \in F$ is split into four parts: First, it is split with respect to $\langle R\rangle\, w_1^{\equiv}$, and then again with respect to $\langle R\rangle\, w_2^{\equiv}$.

In this split of $w_0^{\equiv}$, either the last or the first three parts must be empty: since $w_0^{\equiv}$ is stable with respect to C, either $w_0^{\equiv} \subseteq \langle R\rangle\, w^{\equiv}$ or $w_0^{\equiv} \cap \langle R\rangle\, w^{\equiv} = \{\}$ for all R. If $w_0^{\equiv} \subseteq \langle R\rangle\, w^{\equiv}$, then $(w_0^{\equiv} \setminus \langle R\rangle\, w_1^{\equiv}) \setminus \langle R\rangle\, w_2^{\equiv} = \{\}$. If $w_0^{\equiv} \cap \langle R\rangle\, w^{\equiv} = \{\}$, then both $w_0^{\equiv} \cap \langle R\rangle\, w_1^{\equiv} = \{\}$ and $w_0^{\equiv} \cap \langle R\rangle\, w_2^{\equiv} = \{\}$: since $w^{\equiv} = w_1^{\equiv} \cup w_2^{\equiv}$, it holds that $\langle R\rangle\, w^{\equiv} = \langle R\rangle\, w_1^{\equiv} \cup \langle R\rangle\, w_2^{\equiv}$.

The overall complexity of the algorithm is $O(m \cdot \log n)$, where n is the number of points in the original model, and m is the number of points (partitions) in the result.

6. Completeness

Logicians are interested in logical truths, i.e., in the set of formulas which are valid in *all* models of the logic. How does it help to know about the set of *all* valid formulas when we want to find out whether a particular formula φ holds for a given model or theory? The answer is to encode the model or theory as a set of *assumptions* Φ and check whether the formula in question *follows* from Φ.

In fact, a *logic* can be defined to be any set of well-formed formulas which is closed under provable consequence; and a *theory* is a set of well-formed formulas which is closed under semantical consequence.

Thus there are three notions of consequence involved here:

function `Bisimulation_minimize` (Model $(U, \mathcal{I}, v)$) : Model =
  $C := \{\{U\}\}$, $F := \{\{U\}\}$
  for all $\mathtt{p} \in \mathcal{P}$ and $w^{\equiv} \in F$ **do**
    $F := (F \setminus \{w^{\equiv}\}) \cup \{w^{\equiv} \cap \mathcal{I}(\mathtt{p}),\ w^{\equiv} \setminus \mathcal{I}(\mathtt{p})\}$;
  for all $R \in \mathcal{R}$ and $w^{\equiv} \in F$ **do**
    $F := (F \setminus \{w^{\equiv}\}) \cup \{w^{\equiv} \cap \langle R\rangle\{U\},\ w^{\equiv} \setminus \langle R\rangle\{U\}\}$;
  while $C \neq F$ **do**
    choose $w^{\equiv} \in C \setminus F$ and $w_1^{\equiv} \in F$ such that $w_1^{\equiv} \subseteq w^{\equiv}$
    $w_2^{\equiv} := w^{\equiv} \setminus w_1^{\equiv}$; $C := (C \setminus \{w^{\equiv}\}) \cup \{w_1^{\equiv},\ w_2^{\equiv}\}$;
    for all $R \in \mathcal{R}$ and $w_0^{\equiv} \in F$ **do**
      $F := F \setminus \{w_0^{\equiv}\} \cup$
        $\{(w_0^{\equiv} \cap \langle R\rangle\, w_1^{\equiv}) \cap \langle R\rangle\, w_2^{\equiv},\ (w_0^{\equiv} \cap \langle R\rangle\, w_1^{\equiv}) \setminus \langle R\rangle\, w_2^{\equiv},$
        $(w_0^{\equiv} \setminus \langle R\rangle\, w_1^{\equiv}) \cap \langle R\rangle\, w_2^{\equiv},\ (w_0^{\equiv} \setminus \langle R\rangle\, w_1^{\equiv}) \setminus \langle R\rangle\, w_2^{\equiv}\ \}$
  end;
  return $(F, \mathcal{I}^{\equiv}, v^{\equiv})$

Figure 11: Paige-Tarjan algorithm for bisimulation minimization

- $\Phi \mid\!\vdash \varphi$ if φ follows from Φ,
  i.e. if any model in which all formulas from Φ are valid also validates φ,
- $\Phi \vdash \varphi$ if φ can be proved from Φ,
  i.e. if there is a proof of φ which uses only assumptions from Φ, and
- $(\Phi \rightarrow \varphi)$ if φ is implied by Φ.
  This is a statement of the object language which is only defined if Φ is a single formula. To be liberal, we can identify a finite set of formulas $\Phi \triangleq \{\varphi_1, ..., \varphi_n\}$ with the conjunction $\hat{\Phi} \triangleq (\varphi_1 \wedge ... \wedge \varphi_n)$.

Note that $\Phi \mid\!\vdash \varphi$ is different from $\mathcal{M} \models \varphi$. The notations $\mid\!\vdash \varphi$ and $\vdash \varphi$ are short for $\{\} \mid\!\vdash \varphi$ and $\{\} \vdash \varphi$, respectively.

Of course, the semantical notion of *validity* sometimes is restricted to certain classes of models, e.g., to those satisfying certain axioms, or to natural or tree models.

Also, the algorithmic notion of *provability* sometimes is parameterized by a certain proof-system. In this section, we will use *Hilbert-style* proof-systems, consisting of a set of *axioms* and *derivation rules*. Although such proof systems are not very practical, often they can illustrate the principles underlying completeness proofs. Usually, axioms and derivation rules contain *proposition variables* $q \in \mathcal{Q}$ and a substitution rule allowing consistent replacement of proposition variables with formulas. Conceptually, proposition variables are not the same as propositions, though many authors do not distinguish between these syntactic categories. A free proposition variable in an axiom can be thought of more or less as if it were universally quantified.

To complicate things even more, there are two notions of validity of a formula: *local validity* $(U, \mathcal{I}, w_0) \models \varphi$ (in a model, where the evaluation point is given), and *universal validity* $(U, \mathcal{I}) \models \varphi$ in a frame $(U, \mathcal{I})$. Traditionally, focus has been on complete axiom systems for universal validity rather than for the local version; proofs are much simpler. Thus, in this section we are interested in formulas which are valid *in all models at all points*.

One of the major concerns after defining a logical language and its models is to find an *adequate* proof-system for the logic, i.e. one which is both *sound* and *complete*. That is, for any Φ and φ,

- if $\Phi \vdash \varphi$, then $\Phi \Vdash \varphi$ (Soundness), and
- if $\Phi \Vdash \varphi$, then $\Phi \vdash \varphi$ (Completeness).

It is obvious that any proof system should be sound: we don't want to be able to "prove" false statements. Usually is very easy to prove soundness. We just have to show that the axioms are valid, and that all formulas which can be deduced from valid formulas by the derivation rules are valid. Completeness is often much harder to show, if not impossible. However, it is important to strive for completeness. Firstly, we would like to make sure that any specification which is satisfied by a program can be proved from the program axioms, provided the specification is expressible in the logic. Secondly, and more important, in many cases decision algorithms for automated verification can be obtained from the completeness proofs or vice versa.

6.1. Deductions in Multimodal Logic

To illustrate the basic idea, we start with a simple deductive system for multimodal logic. A number of similar proofs can be found in [Burgess 1984]. We use the following axioms and rules:

(taut)	propositional tautologies
(MP)	$p,\ (p \to q) \vdash q$
(N)	$q \vdash [R]\, q$
(K)	$\vdash ([R](p \to q) \to ([R]\, p \to [R]\, q))$

Since this axiom system is based on the $[R]$-operator rather than the $\langle R \rangle$-operator, we identify $\langle R \rangle\, \varphi$ with $\neg\, [R] \neg \varphi$.

To prove $\Phi \vdash \varphi$ we have to give a *derivation* of φ from the assumptions Φ, i.e., a sequence of formulas such that the last element of this sequence is φ, and every element of this sequence is either from Φ, or a substitution instance of an axiom, or the substitution instance of the consequence of a rule, where all premisses of the rule for this substitution appear already in the derivation.

As an example, let us assume $(\mathtt{p} \to \mathtt{q})$ and derive some consequences:

1.	$(\mathtt{p} \to \mathtt{q})$	(assumption)
2.	$[R](\mathtt{p} \to \mathtt{q})$	(1, **N**)
3.	$([R](\mathtt{p} \to \mathtt{q}) \to ([R]\,\mathtt{p} \to [R]\,\mathtt{q}))$	(**K**)
4.	$([R]\,\mathtt{p} \to [R]\,\mathtt{q})$	(2, 3, MP)

5.	$(\neg \mathtt{q} \to \neg \mathtt{p})$	(1, taut)
6.	$([R]\neg \mathtt{q} \to [R]\neg \mathtt{p})$	(5, as in 1-4)
7.	$(\neg [R]\neg \mathtt{p} \to \neg [R]\neg \mathtt{q})$	(6, taut)
8.	$(\langle R\rangle \mathtt{p} \to \langle R\rangle \mathtt{q})$	(7, $\langle R\rangle \varphi \triangleq \neg [R]\neg\varphi$)

Lines (4) and (8) form the basis for an inductive proof of the following replacement and monotonicity rules:

(repl) $(p \leftrightarrow q) \vdash (\varphi(p) \leftrightarrow \varphi(q))$, and
(mon) $(p \to q) \vdash (\varphi(p) \to \varphi(q))$, where $\varphi(q)$ is positive in q.

(mon) is a syntactical analog of Lemma 3.8. The requirement that $\varphi(q)$ is positive in q means that every occurrence of q is under an even number of negation signs (cf. the definition on Page 1662). For example, $[R]\,q$, $\langle R\rangle\,q$, and $(q \wedge [R](q \vee \langle R\rangle\,q))$ are positive in q.

6.1. THEOREM (Soundness of **ML** axiom system). *If* $\Phi \vdash \varphi$ *then* $\Phi \mid\!\Vdash \varphi$.

PROOF: Soundness of **(taut)** and **(MP)** is immediate. **(N)** is the so called *necessitation rule.* Its validity depends on the universal interpretation of validity: f some formula is valid in every point of a model, it is valid in every point which is the R-successor of some other point in that model. **(K)** is the classical *Kripke*-axiom which holds for all normal modal logics. If in all accessible points p holds, and in all accessible points $(p \to q)$ holds, then in all accessible points q must hold. □

The classical way to prove this theorem is the so–called Henkin-Hasenjäger construction. A set Ψ of formulas is *consistent with* Φ, if there is no finite subset $\{\psi_1, ..., \psi_n\} \subseteq \Psi$ such that $\Phi \vdash (\psi_1 \wedge ... \wedge \psi_n \to \bot)$. Given a set Φ of assumptions and a formula φ which is consistent with Φ, we will construct a model in which Φ is universally valid and φ is locally valid. Call a set w of formulas *maximal*, if for any formula ψ, either ψ or $\neg\psi$ is in w.

6.2. LEMMA (Lindenbaum's extension lemma). *For any formula* φ *which is consistent with* Φ *there exists a maximal consistent set* w_0 *such that* $\varphi \in w_0$ *and* $\Phi \subseteq w_0$

PROOF: Start with $\Phi \cup \{\varphi\}$; for every formula ψ according to a fixed enumeration add either ψ or $\neg\psi$ to w, whichever is consistent with the set constructed so far.□

The *canonical model* for Φ is $(U, \mathcal{I}, w)$, where

- U is the set of maximal consistent sets which include Φ,
- $\mathcal{I}(R) \triangleq \{(w_0, w_1) \mid q \in w_1 \text{ implies } \langle R\rangle\,q \in w_0\}$, and
- $\mathcal{I}(\mathtt{p}) \triangleq \{w_0 \mid \mathtt{p} \in w_0\}$, and
- w is any element from U such that $\varphi \in w$.

The following result is sometimes called the "truth" lemma. Intuitively, it states that any point in the canonical model contains exactly those formulas which are satisfied by this point.

6.3. Lemma (Truth lemma). *Let φ be any formula and w be a maximal consistent set in the canonical model. Then $\varphi \in w$ iff $(U, \mathcal{I}, w) \models \varphi$.*

Proof: The proof is by induction on the structure of φ. In the inductive step, there is one interesting case. We must show that $\langle R \rangle \varphi \in w_0$ iff $(U, \mathcal{I}, w_0) \models \langle R \rangle \varphi$. We first prove that $(U, \mathcal{I}, w_0) \models \langle R \rangle \varphi$ implies $\langle R \rangle \varphi \in w_0$. Since $w_0 \models \langle R \rangle \varphi$, there exists a w_1 such that $w_0 R w_1$ and $w_1 \models \varphi$. By definition of R, we have $\langle R \rangle q \in w_0$ for all $q \in w_1$. Since $w_1 \models \varphi$, the induction hypothesis implies $\varphi \in w_1$. Consequently, $\langle R \rangle \varphi \in w_0$.

For the other direction, assume that $\langle R \rangle \varphi \in w_0$. We have to show that there exists a maximal consistent set w_1 such that $(w_0, w_1) \in \mathcal{I}(R)$ and $\varphi \in w_1$. First observe that the formula $\vdash ((\langle R \rangle \varphi \wedge [R] \psi) \to \langle R \rangle (\varphi \wedge \psi))$ is derivable:

1.	$[R](\psi \to \neg\varphi) \to ([R]\psi \to [R]\neg\varphi)$	(**K**)
2.	$(\neg [R] \neg\varphi \wedge [R]\psi) \to \neg [R](\psi \to \neg\varphi))$	(1, taut)
3.	$(\langle R \rangle \varphi \wedge [R]\psi) \to \langle R \rangle(\varphi \wedge \psi)$	(2, **repl**, taut)

Recall that $[R]\varphi$ is a syntactical abbreviation of $\neg \langle R \rangle \neg\varphi$. In line 3., we replaced $\neg\neg\varphi$ by φ and $\neg(\psi \to \neg\varphi)$ by $(\varphi \wedge \psi)$. This derivation can be generalized to obtain

$$\vdash ((\langle R \rangle \varphi \wedge [R]\psi_1 \wedge \cdots \wedge [R]\psi_n) \to \langle R \rangle(\varphi \wedge \psi_1 \wedge \cdots \psi_n))$$

Because of this result, the set $\{\varphi\} \cup \{\psi \mid [R]\psi \in w_0\}$ must be consistent with Φ. Otherwise, by the definition of consistency on page 1692, there would exist a finite set $\{\psi_1, ..., \psi_n\}$ of formulas such that $[R]\psi_i \in w_0$ for all $1 \leq i \leq n$, and $(\varphi \wedge \psi_1 \wedge \cdots \psi_n \to \bot)$ must be derivable from Φ. Since $\vdash (\langle R \rangle \bot \to \bot)$, we would have $\Phi \vdash (\langle R \rangle(\varphi \wedge \psi_1 \wedge \cdots \psi_n) \to \bot)$. Therefore, $\Phi \vdash ((\langle R \rangle \varphi \wedge [R]\psi_1 \wedge \cdots [R]\psi_n) \to \bot)$. Since $\{\langle R \rangle \varphi, [R]\psi_1, ..., [R]\psi_n\} \subseteq w_0$, the set w_0 would be inconsistent with Φ, which is a contradiction.

Since $\{\varphi\} \cup \{\psi \mid [R]\psi \in w_0\}$ is consistent with Φ, there exists some maximal consistent extension w_1 of this set. Moreover, if $\psi \in w_1$, then $[R]\neg\psi$ can not be in w_0 (otherwise, both ψ and $\neg\psi$ would be in w_1). Since w_0 is maximal, $\psi \in w_1$ implies $\neg [R] \neg\psi = \langle R \rangle \psi \in w_0$. From the definition of $\mathcal{I}(R)$, it follows that $(w_0, w_1) \in \mathcal{I}(R)$. Since $\varphi \in w_1$, the induction hypothesis gives $(U, \mathcal{I}, w_1) \models \varphi$. Together with $(w_0, w_1) \in \mathcal{I}(R)$ we have $(U, \mathcal{I}, w_0) \models \langle R \rangle \varphi$, which was to be shown. □

6.4. Lemma (Satisfiability of consistent formulas). *Every multimodal formula φ consistent with Φ is satisfiable in some model validating Φ.*

Proof: Since for the canonical model $(U, \mathcal{I}, w)$ it holds that $\Phi \subseteq w$ and $\varphi \in w$, Lemma 6.3 asserts that $(U, \mathcal{I}, w) \models \Phi$ and $(U, \mathcal{I}, w) \models \varphi$. Thus every consistent formula is satisfied in its canonical model. □

6.5. Theorem (Completeness). *The deductive system for* **ML** *is complete:*

If $\Phi \mid\!\models \varphi$ then $\Phi \vdash \varphi$.

PROOF: Without loss of generality, we can assume Φ to be consistent with itself: if Φ is inconsistent, then $\Phi \vdash \varphi$ holds trivially. If $\Phi \models \varphi$, then no model in which Φ is universally valid contains a point which satisfies $\{\neg\varphi\}$; therefore with 6.4 it follows that $\{\neg\varphi\}$ is inconsistent with Φ, hence $\Phi \vdash (\neg\varphi \rightarrow \bot)$, which is $\Phi \vdash \varphi$. $\square$

We now show how this proof can be extended for natural models. Recall that a model is called *deterministic*, if all accessibility relations $R \in \mathcal{R}$ are *univalent*: for any given point w there is at most one R-successor of w. The following axiom describes this property.

(U) $\vdash (\langle R\rangle\, q \rightarrow [R]\, q)$

Soundness of this axiom in deterministic models is immediate: if there is any R-successor satisfying q, then all R-successors must satisfy q. In the completeness proof, axiom **U** forces the canonical model to be deterministic: for every $w_0 \in U$ of the canonical model and every $R \in \mathcal{R}$ there can be at most one w_1 with $(w_0, w_1) \in \mathcal{I}(R)$. To see why this is true, assume for contradiction that $(w_0, w_1) \in \mathcal{I}(R)$ and $(w_0, w_1') \in \mathcal{I}(R)$. If $w_1 \neq w_1'$, then there must be a formula ψ such that $\psi \in w_1$ and $\neg\psi \in w_1'$. Therefore $\langle R\rangle\, \psi \in w_0$ and $\langle R\rangle \neg\psi \in w_0$. This is a contradiction to the consistency of w_0: from axiom **U** it follows that if $\langle R\rangle\, \psi \in w_0$, then $\neg \langle R\rangle \neg\psi \in w_0$, since maximal consistent sets are closed under modus ponens. Thus, $\langle R\rangle \neg\psi \notin w_0$. Therefore, we have shown

6.6. THEOREM. **(U)** *is sound and complete for deterministic models.*

There are a number of other axioms which impose specific conditions on the canonical model. To investigate such connections is the topic of *correspondence theory*, see [van Benthem 1984]. Correspondences between modal axioms and relation algebraic expressions can be found in [Schlingloff and Heinle 1997]. (Such an expression is built from basic relation symbols with union, complement, concatenation, and transitive closure.)

As an example for the use of axiom **(U)** in verification, we prove

$$\{(\mathtt{on} \rightarrow \langle R\rangle \neg\mathtt{on} \ \wedge [S]\bot), \quad (\neg\mathtt{on} \rightarrow \langle S\rangle\, \mathtt{on} \ \wedge \langle R\rangle \neg\mathtt{on})\} \quad \vdash \quad [R]\langle S\rangle [S]\bot.$$

The assumptions can be seen as describing the actions of a semaphore with two states, on and ¬on, which can be set with an S-operation when it is not on, and can be reset with an R-operation at any time. The semaphore cannot be set when it is in state on. We want to show that after a reset it is possible to set the semaphore once and only once; that is, for all points reachable with an R operation there exists an S successor from which no further S operation is possible.

1.	$\mathtt{on} \rightarrow \langle R\rangle \neg\mathtt{on} \wedge [S]\bot$	(assuumption)
2.	$\neg\mathtt{on} \rightarrow \langle S\rangle\, \mathtt{on} \wedge \langle R\rangle \neg\mathtt{on}$	(assumption)
3.	$\mathtt{on} \rightarrow \langle R\rangle \neg\mathtt{on}$	(1, taut)
4.	$\neg\mathtt{on} \rightarrow \langle R\rangle \neg\mathtt{on}$	(2, taut)
5.	$\langle R\rangle \neg\mathtt{on}$	(3, 4, taut)

6.	$\langle R \rangle \neg\mathtt{on} \rightarrow [R]\, \neg\mathtt{on}$	(U)
7.	$[R]\, \neg\mathtt{on}$	(5, 6, MP)
8.	$\neg\mathtt{on} \rightarrow \langle S \rangle\, \mathtt{on}$	(2, taut)
9.	$[R]\, \neg\mathtt{on} \rightarrow [R]\, \langle S \rangle\, \mathtt{on}$	(8, mon)
10.	$[R]\, \langle S \rangle\, \mathtt{on}$	(7, 9, MP)
11.	$\mathtt{on} \rightarrow [S]\, \bot$	(1, taut)
12.	$[R]\, \langle S \rangle\, \mathtt{on} \rightarrow [R]\, \langle S \rangle\, [S]\, \bot$	(11, mon)
13.	$[R]\, \langle S \rangle\, [S]\, \bot$	(10, 12, MP)

As we see, even in such simple examples it can be quite difficult to find a Hilbert-style proof "by hand"; therefore it is important to develop automatic proof methods. Algorithms for this purpose are the topic of Section 7.

Consider the case that the logic contains only one accessibility relation ($\mathcal{R} = \{R\}$). Then each path through a deterministic canonical model forms a natural model: let the formula φ be consistent with all substitution instances of the axiom **(U)**. Consider a sequence $\sigma \triangleq (w_0, w_1, w_2, ...)$ of points in the (deterministic) canonical model for φ such that $\varphi \in w_0$ and $w_i R w_{i+1}$ for all i. Obviously, σ is a natural model which initially satisfies φ. Therefore, with axiom **(U)** each consistent formula is satisfiable in a natural model; in other words, **(U)** is complete for monomodal logic in natural models. The same holds if we require univalence of the transition relation $\prec \triangleq \bigcup \mathcal{R}$:

(N) $q \vdash \cancel{\mathbf{X}}\, q$
(K) $\vdash (\cancel{\mathbf{X}}(p \rightarrow q) \rightarrow (\cancel{\mathbf{X}}\, p \rightarrow \cancel{\mathbf{X}}\, q))$
(U) $\vdash (\mathbf{X}\, q \rightarrow \cancel{\mathbf{X}}\, q)$

Together with **(taut)** and **(MP)**, these axioms are sound and complete for the **X**-operator in natural models.

6.2. Transitive Closure Operators

A major difference between temporal and modal logic is that temporal logic has operators for the transitive closure of the transition relation. In order to motivate the discussion in the completeness proofs for **CTL** and **LTL**, in this subsection we extend the above completeness proof to handle such operators. For simplicity, we first give the proof for the logic with operators **X** (or, equivalently **E X**) for the transition relation and $\mathbf{F}^*$ (or $\mathbf{E}\,\mathbf{F}^*$) for its reflexive transitive closure (plus derived operators $\cancel{\mathbf{X}}\, \varphi \triangleq \neg\, \mathbf{X} \neg\varphi$, $\mathbf{G}^*\, \varphi \triangleq \neg\, \mathbf{F}^* \neg\varphi$, etc.). The necessary generalizations for **CTL** and **LTL** are indicated at the end of this subsection.

Close inspection of the semantics of $\mathbf{F}^*$ reveals a fundamental problem, compared to the completeness proof given above. Consider the set $\Phi \triangleq \{\mathtt{p}, \cancel{\mathbf{X}}\mathtt{p}, \cancel{\mathbf{X}}\cancel{\mathbf{X}}\mathtt{p}, \cancel{\mathbf{X}}\cancel{\mathbf{X}}\cancel{\mathbf{X}}\mathtt{p}, ... \}$. Then clearly $\Phi \Vdash \mathbf{G}^*\, \varphi$. However, $\Phi \not\vdash \mathbf{G}^*\, \varphi$, since every proof of $\mathbf{G}^*\, \varphi$ from Φ can use only a limited number of premisses (proofs are *finite* sequences). But there does not exist a finite subset $\Phi_0 \subset \Phi$ such that the statement $\Phi_0 \vdash \mathbf{G}^*\, \varphi$ holds.

Thus, the above completeness proof fails. For an arbitrary set Φ, it may not be possible to construct a maximal consistent extension, since we can not apply an axiom to show the consistency of an infinite set of premisses.

When dealing with second order concepts like transitive closure we have to limit ourselves to a weaker form of completeness. An axiom system is called *weakly complete*, if $\Phi \mathrel{||\!\!-} \varphi$ implies $\Phi \vdash \varphi$ for all *finite* Φ.

In first order logic, the *deduction theorem* makes it possible to discard any finite set of assumptions: $\psi \mathrel{||\!\!-} \varphi$ iff $\mathrel{||\!\!-} (\forall\psi \to \varphi)$, where $\forall\psi$ is the universal closure of ψ. In temporal logic, a similar deduction theorem holds:

6.7. THEOREM (Deduction theorem). *$\psi \mathrel{||\!\!-} \varphi$ iff $\mathrel{||\!\!-} (\mathbf{G}^* \psi \to \varphi)$.*

Therefore, to prove weak completeness it is sufficient to prove that $\mathrel{||\!\!-} \varphi$ implies $\vdash \varphi$. We use the following axiom system (in addition to modus ponens **(MP)** and propositional tautologies **(taut)**):

(N)	$q \vdash \bar{\mathbf{X}}\, q$
(K)	$\vdash (\bar{\mathbf{X}}(p \to q) \to (\bar{\mathbf{X}}\, p \to \bar{\mathbf{X}}\, q))$
(Rec)	$\vdash (\mathbf{G}^* q \to (q \wedge \bar{\mathbf{X}}\, \mathbf{G}^* q))$
(Ind)	$(p \to (q \wedge \bar{\mathbf{X}}\, p)) \vdash (p \to \mathbf{G}^* q)$

Dually, the last axiom and rule can be written as

(Rec)	$\vdash ((q \vee \mathbf{X}\, \mathbf{F}^* q) \to \mathbf{F}^* q)$
(Ind)	$((q \vee \mathbf{X}\, p) \to p) \vdash (\mathbf{F}^* q \to p)$

(N) and **(K)** are "nexttime-versions" of the respective modal rule and axiom given above. In this subsection, we prove completeness for general Kripke structures (with a possibly nondeterministic accessibility relation), thus there is no need for the temporal version of **(U)**. Axiom **(Rec)** and rule **(Ind)** are sometimes attributed to Segerberg. They reflect the definition of the transitive closure as the minimal transitive relation which includes all accessibility relations. **(Rec)** is the *recursion* axiom which can be used to unfold a $\mathbf{G}^*$-operator (cf. Subsection 3.2, Page 1663):

$$\mathbf{G}^* \varphi \to (\varphi \wedge \bar{\mathbf{X}}(\varphi \wedge \bar{\mathbf{X}}(\varphi \wedge \ldots))).$$

(Ind) is the *induction* rule which can be used to deduce a property $\mathbf{G}^* \varphi$ from an *invariant* ψ, i.e., from a formula ψ for which $(\psi \to \bar{\mathbf{X}}\, \psi)$ and $(\psi \to \varphi)$ are derivable.

6.8. LEMMA. **(Rec)** *and* **(Ind)** *are sound: $\vdash \varphi$ implies $\models \varphi$.*

For the soundness of **(Rec)**, observe that $w \models \mathbf{G}^* q$ means that for all $u \geq w$ it holds that $u \models q$. Thus $w \models q$, and for all $v \succ w$ and $u \geq v$ we have $u \models q$, which means $w \models \bar{\mathbf{X}}\, \mathbf{G}^* \varphi$.

For the soundness of **(Ind)**, assume that $(p \to (q \wedge \bar{\mathbf{X}}\, p))$ is universally valid in a frame $\mathcal{F} \triangleq (U, \mathcal{I})$, that is, for any $w \in U$, if $w \models p$, then $w \models q$ and $v \models p$ for all $v \succ w$. Assume further that $w_0 \models p$, and show that $w \models q$ for all $w \geq w_0$. We show that $w \models p$ for all $w \geq w_0$. From this the claim follows since $w \models p$ implies $w \models q$. The proof is by induction on the length of the shortest path between w_0 and w. If this length is zero, then $w_0 = w$, and there is nothing to show. For the inductive step, assume that the shortest path from w_0 to w has $n+1$ elements. Then there exists a predecessor $w' \prec w$ such that $w_0 \leq w'$, and the shortest path between w_0 and w' has n elements. From the induction hypothesis, $w' \models p$. Since $w' \prec w$, it follows that $w \models p$. □

Next, we show that these axioms are complete for transitive closure. Up to the truth lemma, the proof is almost the same as for modal logic. But, we only use *finite* maximal consistent sets: we start with a single (finite) consistent formula φ for which we have to construct a model. The set $ESF(\varphi)$ of *extended sub-formulas* of φ (sometimes also called *Fischer-Ladner closure*, [Fischer and Ladner 1979]) is the following set of formulas:

- φ_1 and φ_2 are extended sub-formulas of $(\varphi_1 \to \varphi_2)$, (thus φ is an extended sub-formula of $\neg\varphi$)
- φ is an extended sub-formula of $\mathbf{X}\,\varphi$,
- φ and $\mathbf{X}\,\mathbf{F}^*\,\varphi$ are extended sub-formulas of $\mathbf{F}^*\,\varphi$,
- φ is an extended sub-formula of φ, and
- every extended sub-formula of an extended sub-formula of φ is an extended sub-formula of φ.

For any given φ, the set $ESF(\varphi)$ is finite. A consistent set of formulas is called *finitely maximal*, if it is maximal with respect to $ESF(\varphi)$; that is, for every extended sub-formula ψ of φ, either ψ or $\neg\psi$ is in the finitely maximal consistent set.

As in the infinite case, for any consistent formula φ there exists at least one consistent set w_0 which is finitely maximal with respect to $ESF(\varphi)$ such that $\varphi \in w_0$. Consider the following *finite canonical model* $(U, \mathcal{I}, w)$:

- U is the set of finitely maximal consistent sets,
- $\mathcal{I}(\prec) \triangleq \{(w_0, w_1) \mid \neg\,\mathbf{X}\,q \in w_0 \text{ implies } \neg q \in w_1\}$, and
- $\mathcal{I}(\mathtt{p}) \triangleq \{w_0 \mid \mathtt{p} \in w_0\}$, and
- w is any element from U such that $\varphi \in w$.

Compare this with the canonical model for modal logic on Page 1692. Similar as in Lemma 6.3, for any extended sub-formula φ and finitely maximal consistent set w, the following statement holds:

6.9. LEMMA (Truth lemma for transitive closure operators). $w \models \varphi$ *iff* $\varphi \in w$.

From this truth lemma, completeness follows exactly as in the multimodal case.
PROOF: The proof is by induction on φ. The case $\varphi = \mathbf{X}\,\psi$ is proven almost exactly as in the completeness proof for modal logic. If $(U, \mathcal{I}, w_0) \models \mathbf{X}\,\psi$, then there exists a w_1 such that $w_0 \prec w_1$ and $w_1 \models \psi$. Assuming for contradiction that $\mathbf{X}\,\psi \notin w_0$, we have $\neg\,\mathbf{X}\,\psi \in w_0$, since the set of extended sub-formulas is closed under (single) negation. From the definition of $\mathcal{I}(\prec)$ we can infer that $\neg\psi \in w_1$, i.e., $\psi \notin w_1$. According to the induction hypothesis, $w_1 \not\models \psi$, which is a contradiction. In the other direction, assume that $\mathbf{X}\,\psi \in w_0$, and let w_1 be any finitely maximal

consistent extension of $\{\psi\} \cup \{\neg\xi \mid \neg\,\mathbf{X}\,\xi \in w_0\}$. Since $\psi \in w_1$, the induction hypothesis gives $(U, \mathcal{I}, w_1) \models \psi$. According to the definition of $\mathcal{I}(\prec)$ it holds that $w_0 \prec w_1$. Therefore $(U, \mathcal{I}, w_0) \models \mathbf{X}\,\psi$.

Thus, it remains to show that $\mathbf{F}^{*}\psi \in w_0$ iff $(U, \mathcal{I}, w_0) \models \mathbf{F}^{*}\psi$. For one direction, assume that $\mathbf{F}^{*}\psi \notin w_0$. We have to prove that $w_0 \not\models \mathbf{F}^{*}\psi$. In other words, if $w_0 \leq w_n$ then it has to be shown that $w_n \not\models \psi$. Note that $w_0 \leq w_n$ iff there is a finite path $(w_0, w_1, ..., w_n)$ such that $w_i \prec w_{i+1}$ for all $i < n$. We show by induction on n that $\neg\,\mathbf{F}^{*}\psi \in w_n$, hence $\mathbf{F}^{*}\psi \notin w_n$. For $n = 0$, there is nothing to show. For $n > 0$, the induction hypothesis guarantees that $\mathbf{F}^{*}\psi \notin w_{n-1}$, i.e., $\neg\,\mathbf{F}^{*}\psi \in w_{n-1}$. Both $\mathbf{X}\,\mathbf{F}^{*}\psi$ and $\neg\,\mathbf{X}\,\mathbf{F}^{*}\psi$ are extended sub-formulas of $\mathbf{F}^{*}\varphi$, therefore one of them must be in w_{n-1}. From axiom **(Rec)**, the formula $(\neg\,\mathbf{F}^{*}\psi \to \neg\,\mathbf{X}\,\mathbf{F}^{*}\psi)$ can be derived. Consequently, $\neg\,\mathbf{X}\,\mathbf{F}^{*}\psi \in w_{n-1}$. Thus by the definition of $\mathcal{I}(\prec)$, we have $\neg\,\mathbf{F}^{*}\psi \in w_n$. Now we show that $w_n \not\models \psi$. Since axiom **(Rec)** derives $(\neg\,\mathbf{F}^{*}\psi \to \neg\psi)$ and $\neg\,\mathbf{F}^{*}\psi \in w_n$, the assumption $\psi \in w_n$ would contradict the consistency of w_n. Therefore, $\psi \notin w_n$. According to the induction hypothesis, $w_n \not\models \psi$.

Now we prove that $\mathbf{F}^{*}\psi \in w_0$ implies $w_0 \models \mathbf{F}^{*}\psi$. For any finitely maximal consistent set w and any (finite) set W of such sets, let $\hat{w} \triangleq \bigwedge\{\psi \mid \psi \in w\}$, and $\check{W} \triangleq \bigvee\{\hat{w} \mid w \in W\}$. Furthermore, let $X_w \triangleq \{w' \mid w \prec w'\}$. An important step is to prove

$$(*) \qquad\qquad \vdash (\hat{w} \to \cancel{\mathbf{X}}\,\check{X}_w)$$

Since $\vdash ((\cancel{\mathbf{X}}\,\psi_1 \wedge \cancel{\mathbf{X}}\,\psi_2) \to \cancel{\mathbf{X}}(\psi_1 \wedge \psi_2))$, we can infer $\vdash (\bigwedge\{\cancel{\mathbf{X}}\,\psi_i\} \to \cancel{\mathbf{X}}\bigwedge\{\psi_i\})$. Therefore, $\vdash (\hat{w} \to \cancel{\mathbf{X}}\bigwedge\{\neg q \mid \neg\,\mathbf{X}\,q \in w\})$. Since U is the set of *all* finitely maximal consistent sets, $\vdash \check{U}$ can be proven by propositional reasoning: for each φ and $\mathtt{p}$, it is valid that $\varphi \vdash ((\varphi \wedge \mathtt{p}) \vee (\varphi \wedge \neg\mathtt{p})$. Since $\check{U}$ is the disjunction of all possible conjunction of positive and negative literals from $\mathcal{P}$, it is derivable from this formula. Therefore, $\vdash (\hat{w} \to \cancel{\mathbf{X}}\,\check{U})$. Together, this gives $\vdash (\hat{w} \to \cancel{\mathbf{X}}(\check{U} \wedge \bigwedge\{\neg q \mid \neg\,\mathbf{X}\,q \in w\}))$. Consequently, $\vdash (\hat{w} \to \cancel{\mathbf{X}}\bigvee\{\hat{u} \wedge \bigwedge\{\neg q \mid \neg\,\mathbf{X}\,q \in w\} \mid u \in U\})$. If $w' \triangleq u \cup \{\neg q \mid \neg\,\mathbf{X}\,q \in w\}$ is inconsistent, then $\vdash (\hat{w'} \to \bot)$. If w' is consistent, then $w \prec w'$ according to the definition of $\mathcal{I}(\prec)$, i.e., $w' \in X_w$. Therefore, $\vdash (\hat{w} \to \cancel{\mathbf{X}}\bigvee\{\hat{w'} \mid w' \in X_w\})$, which proves $(*)$.

Since there are only finitely many extended sub-formulas, the universe U is finite. Let $W \triangleq \{w_0, w_1, ..., w_n\}$ be the set $\{w' \in U \mid w_0 \leq w'\}$. From $(*)$, it follows that $\vdash (\bigvee\{\hat{w} \mid w \in W\} \to \bigvee\{\cancel{\mathbf{X}}\,\check{X}_w \mid w \in W\})$. Furthermore, $\vdash (\bigvee\{\cancel{\mathbf{X}}\,\check{X}_w\} \to \cancel{\mathbf{X}}\bigvee\{\check{X}_w\})$. Since $\{X_w \mid w \in W\} \subseteq W$, it holds that $\vdash (\bigvee\{\hat{X_w} \mid w \in W\} \to \check{W})$. Therefore,

$$(**) \qquad\qquad \vdash (\check{W} \to \cancel{\mathbf{X}}\,\check{W})$$

Assume that $w_0 \not\models \mathbf{F}^{*}\psi$ and show that $\mathbf{F}^{*}\psi \notin w_0$. From the assumption, $w \not\models \psi$ for all $w \in W$. As above, the induction hypothesis implies that $\psi \notin w$ for all $w \in W$, i.e., $\neg\psi \in w$. Consequently, $(\hat{w} \to \neg\psi)$ for all $w \in W$, which implies $\vdash (\check{W} \to \neg\psi)$. Together with $(**)$ we have $\vdash (\check{W} \to (\neg\psi \wedge \cancel{\mathbf{X}}\,\check{W}))$. Thus, by **(Ind)**, $\vdash (\check{W} \to \mathbf{G}^{*}\neg\psi)$. Since $w_0 \in W$, it holds that $\vdash (\hat{w_0} \to \check{W})$. Therefore, $\vdash (\hat{w_0} \to \neg\,\mathbf{F}^{*}\psi)$. Since w_0 is consistent, $\mathbf{F}^{*}\psi \notin w_0$. $\square$

6.10. LEMMA. *(**(N)**, **(K)**, **(Rec)**, **(Ind)**) is complete: if* $\models \varphi$ *then* $\vdash \varphi$.

PROOF: The theorem follows from Lemma 6.9 similar as Theorem 6.5 follows from Lemma 6.3 for multimodal logic. □

This completeness proof can easily be extended to **CTL** [Emerson and Halpern 1985]. The following axiom system (in addition to propositional logic) is sound and complete:

$$
\begin{array}{ll}
\textbf{(N)} & q \vdash \mathbf{A}\,\cancel{\mathbf{X}}\, q \\
\textbf{(K)} & \vdash (\mathbf{A}\,\cancel{\mathbf{X}}(p \rightarrow q) \rightarrow (\mathbf{A}\,\cancel{\mathbf{X}}\, p \rightarrow \mathbf{A}\,\cancel{\mathbf{X}}\, q)) \\
\textbf{(RecEU}^{+}\textbf{)} & \vdash (\mathbf{E}\,\mathbf{X}(q_2 \vee q_1 \wedge \mathbf{E}(q_1\, \mathbf{U}^{+}\, q_2)) \rightarrow \mathbf{E}(q_1\, \mathbf{U}^{+}\, q_2)) \\
\textbf{(RecAU}^{+}\textbf{)} & \vdash (\mathbf{A}\,\mathbf{X}(q_2 \vee q_1 \wedge \mathbf{A}(q_1\, \mathbf{U}^{+}\, q_2)) \rightarrow \mathbf{A}(q_1\, \mathbf{U}^{+}\, q_2)) \\
\textbf{(IndEU}^{+}\textbf{)} & (\mathbf{E}\,\mathbf{X}(q_2 \vee q_1 \wedge p) \rightarrow p) \vdash (\mathbf{E}(q_1\, \mathbf{U}^{+}\, q_2) \rightarrow p) \\
\textbf{(IndAU}^{+}\textbf{)} & (\mathbf{A}\,\mathbf{X}(q_2 \vee q_1 \wedge p) \rightarrow p) \vdash (\mathbf{A}(q_1\, \mathbf{U}^{+}\, q_2) \rightarrow p)
\end{array}
$$

For **LTL**, proving completeness for natural models is more intricate, since we have to construct a natural model from the canonical model. The axiom system for the future fragment uses suitable versions of **(N)**, **(K)**, **(Rec)**, **(Ind)** and **(U)**. For **LTL** with past operators, additional axioms are necessary which describe the relation between $\mathbf{U}^{+}$ and $\mathbf{U}^{-}$. Several elaborate proofs can be found in the literature [Prior 1957, Gabbay et al. 1980, Burgess 1984, Lichtenstein et al. 1985, Kröger 1987].

A sound and complete proof system for **qTL** was described in [Kesten and Pnueli 1995]. We just briefly indicate how the above axioms can be extended for μ**TL**:

$$
\begin{array}{ll}
\textbf{(Rec}\nu\textbf{)} & \vdash (\nu q\ \varphi \rightarrow \varphi\{q := \nu q\ \varphi\}) \\
\textbf{(Ind}\nu\textbf{)} & (p \rightarrow \varphi\{q := p\}) \vdash (p \rightarrow \nu q\ \varphi)
\end{array}
$$

An equivalent formulation which is based on the least fixpoint operator is

$$
\begin{array}{ll}
\textbf{(Rec}\mu\textbf{)} & \vdash (\varphi\{q := \mu q\ \varphi\} \rightarrow \mu q\ \varphi) \\
\textbf{(Ind}\mu\textbf{)} & (\varphi\{q := p\} \rightarrow p) \vdash (\mu q\ \varphi \rightarrow p)
\end{array}
$$

All recursion and induction axioms above can be obtained as special cases of these very general axioms. For their soundness, we refer to the Knaster-Tarski fixpoint properties in Corollary 3.9. The completeness proof can be adapted to show completeness for a certain subclass of positive μ**TL** formulas, the *aconjunctive* ones [Kozen 1983]. This restriction enforces that if $\nu r\ \psi_1$ and $\nu s\ \psi_2$ are subformulas of $\nu q\ \psi$ each containing an occurrence of the same variable q, then no two occurrences of variables r and s are conjunctively related.

The problem of completeness of these axioms for *all* μ**TL** formulas was solved in [Walukiewicz 1995]. It can be shown that for any formula there exists an equivalent aconjunctive formula. Thereby it suffices to derive this aconjunctive formula from the axioms in order to prove any given formula.

In these proofs, there is a pattern which will frequently reappear in subsequent sections. An *invariance* is a negative occurrence of a least fixpoint operator, or a positive occurrence of a greatest fixpoint operator (e.g., $\mathbf{G}^{*}$, $\mathbf{W}^{+}$, ν). Dually, an *eventuality* ($\mathbf{F}^*$, $\mathbf{U}^+$, μ etc.) is a positive occurrence of a least fixpoint operator, or a negative occurrences of greatest fixpoint operator. In the completeness proof, invariances are unfolded via the recursion axiom, whereas eventualities are fulfilled using the recursion axiom.

7. Decision Procedures

In this section we derive *decision procedures* for some of the logics introduced above. As shown by Büchi and Rabin [Büchi 1962, Rabin 1969], monadic second order logic on natural and tree models is decidable. Therefore, all logics which have a validity-preserving standard translation into **MSOL** or **SnS** (second order logic of n successors) are decidable. However, this proof does not yield efficient decision algorithms. In this section, we will develop such algorithms from the completeness proofs of the previous section. Given a set of assumptions Φ and a formula φ, we want to decide whether $\Phi \vdash \varphi$ or not. By completeness, $\Phi \vdash \varphi$ iff $\Phi \Vdash \varphi$. Even though multimodal logic is complete, for arbitrary sets Φ of assumptions and a given formula φ it is not decidable whether $\Phi \Vdash \varphi$. Therefore, we restrict attention to finite sets of assumptions. Hence we need an algorithm which, given a formula φ and a finite set of assumptions Φ, decides whether there is a model which globally validates Φ such that φ is satisfied in the initial point.

If such a model exists, then often the size of the canonical model for Φ and φ can be bounded by a function of the length of the formulas $\hat{\Phi}$ and φ ("finite model property"). Therefore, many propositional modal and temporal logics are decidable: it is sufficient to check all models up to a certain size whether they are appropriate. However, this is not practical. In this section, we show how to construct a model effectively.

There are two main appraches. "Global" algorithms start with the largest possible model and shrink it to an appropriate size. "Local" algorithms start with a minimal model which is extended until it is a model for the formula. For technical reasons, global algorithms seem to be more adequate for the branching time approach, and local algorithms seem to be better suited for linear temporal logics.

7.1. Deciding Branching Time Logics

To decide whether a given multimodal formula φ is satisfiable with assumptions Φ, we try in a systematic way to construct the canonical model for $\Phi \Vdash \varphi$. In the universe of this model, points are maximal consistent sets of formulas. Since we assume that the set Φ of assumptions is finite, it is sufficient to consider maximality with respect to all sub-formulas of $\hat{\Phi}$ and φ. In the following, we assume that φ and Φ are given and write SF for the (finite) set of all of these sub-formulas. We use subsets of SF to represent maximal sets of sub-formulas. That is, a set $w \subseteq SF$ represents the maximal set $\{\psi \mid \psi \in w\} \cup \{\neg\psi \mid \psi \notin w\}$. A set $w \subseteq SF$ of subformulas is called *propositionally consistent*, if

- $\bot \notin w$, and
- if $(\psi_1 \to \psi_2) \in SF$, then $(\psi_1 \to \psi_2) \in w$ iff $\psi_1 \in w$ implies $\psi_2 \in w$.

That is, $(\psi_1 \to \psi_2) \in w$ iff $\psi_1 \notin w$ or $\psi_2 \in w$. Expanding the definitions it can be shown that

- if $\neg\psi \in SF$, then $\neg\psi \in w$ iff $\psi \notin w$,
- if $(\psi_1 \wedge \psi_2) \in SF$, then $(\psi_1 \wedge \psi_2) \in w$ iff $\psi_1 \in w$ and $\psi_2 \in w$, and
- if $(\psi_1 \vee \psi_2) \in SF$, then $(\psi_1 \vee \psi_2) \in w$ iff $\psi_1 \in w$ or $\psi_2 \in w$.

Any propositionally consistent set is "consistent for propositional logic": if we consistently replace any modal formula in w by a new proposition, then the resulting set of formulas is satisfiable in propositional logic. A satisfying interpretation is given by $\mathcal{I}(\mathtt{p}) \triangleq \mathbf{true}$ iff $\mathtt{p} \in w$.

To construct the canonical model of a consistent formula, let the universe U initially be the set of propositionally consistent sets of sub-formulas which contain all assumptions. That is, $U \triangleq \{w \subseteq SF \mid \Phi \subseteq w\}$. The obvious choice for $\mathcal{I}(\mathtt{p})$ then is $\{w \mid \mathtt{p} \in w\}$. The initial interpretation of any $\langle R \rangle$ operator is the universal relation $U \times U$. The decision procedure iteratively deletes 'bad arcs' and 'bad points' until stabilization is reached. *Bad arcs* are pairs $(w_0, w_1) \in \mathcal{I}(R)$ such that w_0 contains $[R]\,\psi$ but it is not the case that $\psi \in w_1$. More precisely, an arc (w_0, w_1) is bad if for some sub-formula $\langle R \rangle\,\psi$ it holds that $\langle R \rangle\,\psi \notin w_0$ and $\psi \in w_1$. *Bad points* w_0 contain a formula $\langle R \rangle\,\psi$, but there does not (or no longer) exist a tuple $(w_0, w_1) \in \mathcal{I}(R)$ with $\psi \in w_1$. If upon termination there is a point w which was not deleted such that $\varphi \in w$, it returns "satisfiable", else it returns "unsatisfiable".

7.1. LEMMA. *The modal logic decision procedure is sound: φ is satisfiable in some model which universally validates Φ iff the procedure returns "satisfiable".*

PROOF: For one direction, let $\mathcal{M} = (U, \mathcal{I}, w_0)$ be the result of the above deletion procedure. That is, assume that $\mathcal{M}$ does not contain a bad arc or bad point, and that $w_0 \in U$ is some point with $\varphi \in w_0$. We show that $(U, \mathcal{I}) \models \Phi$ and $(U, \mathcal{I}, w_0) \models \varphi$. Similar to the truth Lemma 6.3, for every $w \in U$ and every $\psi \in SF$ it holds that $(\psi \in w)$ iff $(U, \mathcal{I}, w) \models \psi$. This is shown by induction on the structure of ψ: for atomic propositions and boolean combinations of formulas the statement is just a consequence of the respective definitions. For modal subformulas, it follows from the deletion rules in the decision procedure: if $[R]\,\psi \in w$, then for all $w' \in U$ such that $(w, w') \in \mathcal{I}(R)$ it must be the case that $\psi \in w'$. This holds since $\mathcal{M}$ does not contain any bad arcs. By the induction hypothesis, $w' \models \psi$, and therefore $w \models [R]\,\psi$. If $\langle R \rangle\,\psi \in w$, then there is some $w' \in U$ such that $(w, w') \in \mathcal{I}(R)$ and $\psi \in w'$. This holds since $\mathcal{M}$ does not contain any bad points. As above, it follows that $w \models \langle R \rangle\,\psi$. Thus, the assumption $\varphi \in w_0$ implies that $w_0 \models \varphi$. Moreover, since every $w \in U$ contains Φ we have shown that φ is satisfiable in a model which globally validates Φ.

For the other direction, assume that for some (finite or infinite) model $\mathcal{M} = (U, \mathcal{I}, w_0)$ it holds that $w_0 \models \varphi$, and $w \models \hat{\Psi}$ for all $w \in U$. We have to show that the above procedure terminates successfully. For any $w \in U$, let $w^{\equiv} \triangleq \{\psi \in SF \mid$

$w \models \psi\}$. Since SF is finite, there are only finitely many such $w^{\equiv}$. Let the *filtration* of $\mathcal{M}$ be $\mathcal{M}^{\equiv} \triangleq (U^{\equiv}, \mathcal{I}^{\equiv}, w_0^{\equiv})$, where

- $U^{\equiv} \triangleq \{w \mid w = u^{\equiv} \text{ for some } u \in U\}$,
- $(w_1, w_2) \in \mathcal{I}^{\equiv}(R)$ iff there are $u_1, u_2 \in U$ such that $w_1 = u_1^{\equiv}$ and $w_2 = u_2^{\equiv}$ and $(u_1, u_2) \in \mathcal{I}(R)$,
- $w \in \mathcal{I}^{\equiv}(\mathtt{p})$ iff $\mathtt{p} \in w$, and
- $w_0^{\equiv} = \{\psi \in SF \mid w_0 \models \psi\}$.

Clearly, $\mathcal{M}^{\equiv}$ is a submodel of the initial model of our decision algorithm. Moreover, no point or arc of $\mathcal{M}^{\equiv}$ is ever removed by the decision procedure. Therefore, the algorithm terminates with a nonempty result. Since $w_0 \models \varphi$, it holds that $\varphi \in w_0^{\equiv}$.$\square$

Since the decision algorithm iterates over all of the points and sub-formulas, there are two ways to implement it. First, we can implement it by a search of all points using nested iteration for all sub-formulas of this point. The second technique is to use a bottom up search of all sub-formulas, where we check all points and arcs to see whether they are 'bad' with respect to this formula. In both cases, it is important to repeat the search after some deletions have taken place, until stabilization is reached. A pseudo-code description is given in Fig. 12. Recall that $R(w)$ denotes the set of successors of point w with respect to relation R. Furthermore, for any set of points U and formula ψ, let U_ψ denote $\{w \in U \mid \psi \in w\}$.

Depending on the data structures used for the representation of sets, it may not be necessary to implement set operations by a traversal of all elements of the set. For example, all set operations which are used in the comment lines of the pseudo-code can be implemented directly with a BDD representation for U and R as described in Section 10.

In a concrete implementation of this algorithm, there is a tradeoff between computation time and space: for any sub-formula $\psi \triangleq (\psi_1 \to \psi_2)$ and any point w, it can be determined whether $\psi \in w$ by deciding whether $\psi_1 \notin w$ or $\psi_2 \in w$. Hence, it is not necessary to represent a propositionally consistent set by the set of sub-formulas it consists of; boolean combinations of sub-formulas can be omitted. A point then is represented by

- the set of sub-formulas which are atomic propositions, and
- the set of sub-formulas which are of the kind $\langle R \rangle \psi$.

If we use this representation, then we may have to calculate the value of boolean combinations of formulas from their constituent parts. This value is needed in order to determine whether the representation of a point is propositionally consistent with the assumptions.

We now show how to extend this algorithm to transitive closure operators. The recursion axiom for the $\mathbf{F}^*$-operator can be written as follows: (cf. page 1696):

(Rec) $\quad \vdash (\neg \mathbf{F}^* q \to \neg q \wedge \mathbf{X} \neg \mathbf{F}^* q)$

This axiom indicates that if $\mathbf{F}^* \psi \notin w_0$, then $\psi \notin w_0$ and for all w_1 such that $w_0 \prec w_1$ it should hold that $\mathbf{F}^* \psi \notin w_1$. Thus, in the model all points for which $\mathbf{F}^* \psi \notin w_0$ and $\psi \in w_0$ have to be deleted. Similarly, 'bad arcs' (w_0, w_1) are those for which $\mathbf{F}^* \psi \notin w_0$ and $\mathbf{F}^* \psi \in w_1$.

```
procedure ML_sat (Formula φ, Formulaset Φ) =
/* Input Φ and φ, determine if φ satisfiable with global assumptions Φ */
    U := {w ⊆ SF | Φ ⊆ w, ⊥ ∉ w};
    /* delete propositionally inconsistent points */
    for all ψ = (ψ1 → ψ2) ∈ SF do
        /* U := U ∩ ((U_ψ \ U_ψ1) ∪ (U_ψ ∩ U_ψ2) ∪ (U_ψ1 \ U_ψ \ U_ψ2)) */
        for all w ∈ U do
            if (ψ ∈ w ∧ ψ1 ∈ w ∧ ψ2 ∉ w) ∨ (ψ ∉ w ∧ (ψ1 ∉ w ∨ ψ2 ∈ w))
            then U := U \ {w};
    R := U × U;
    repeat until stabilization
        for all ψ = ⟨R⟩ψ1 ∈ SF do
            /* delete bad arcs */
            /* R := R ∩ ((U_ψ × U) ∪ (U × (U \ U_ψ1))) */
            for all (w0, w1) ∈ R do
                if w0 ∉ U ∨ w1 ∉ U ∨ (ψ ∉ w0 ∧ ψ1 ∈ w1)
                then R := R \ {(w0, w1)};
            /* delete bad points */
            /* U := (U \ U_ψ) ∪ (U ∩ {w | (R(w) ∩ U_ψ1) ≠ {}}) */
            for all w ∈ U do
                if (ψ ∈ w ∧ ∀w' ∈ R(w) (ψ1 ∉ w')) then U := U \ {w};
    if U_φ = {}
    then print(φ, "is not satisfiable with assumptions", Φ)
    else print(φ, "and the assumptions", Φ, "are satifiable in", U_φ)
```

Figure 12: Modal logic decision algorithm

In modal logic, a 'bad point' was defined to be one which contains $\langle R\rangle\,\psi$, but no R-successor contains ψ. For transitive closure operators, however, it is not sufficient to delete all points w_0 for which $\mathbf{F}^*\,\psi \in w_0$, $\psi \notin w_0$ and no successor contains $\mathbf{F}^*\,\psi$. There might be a closed loop of points all of which contain $\mathbf{F}^*\,\psi$, but no point containing ψ is reachable from the loop. A point is bad, if it *contains* $\mathbf{F}^*\,\psi$, but does not *fulfill* this eventuality, i.e., no reachable point w_n contains ψ. To check this condition, we need another iteration: for each sub-formula of the form $\mathbf{F}^*\,\psi$ we iteratively mark all points which can reach a point containing ψ. We initially mark all points which contain ψ. We then continue to mark all points which have a marked successor. After stabilization all formulas $\mathbf{F}^*\,\psi$ in unmarked points are unsatisfied and the respective points can be deleted. The algorithm, which is an extension of the algorithm in Figure 12, is given in Figure 13. For a correctness proof and an extension to **CTL**, see [Emerson and Sistla 1984, Emerson 1990]

```
for all ψ = F* ψ1 ∈ SF do
    /* delete bad 'arcs' */
    U := U \ {w |  ψ ∉ w0 ∧ ψ1 ∈ w0};
    R := R \ {(w0, w1) |  ψ ∉ w0 ∧ ψ ∈ w1};
    /* mark all points which can reach ψ1 */
    New := {w0 | ∃w1 ∈ U :  (w0, w1) ∈ R ∧ ψ1 ∈ w1};
    Marked := New;
    repeat
        New:= {w0 | ∃w1 ∈ New: (w0, w1) ∈ R}\ Marked;
        Marked := Marked ∪ New;
    until New={};
    /* delete bad points */
    U := U \ {w | ψ1 ∈ w ∧ w ∉ Marked};
```

Figure 13: marking algorithm for transitive closure

7.2. Satisfiability Algorithms for Natural Models

The branching time decision procedures described in the previous subsection construct a "most general" model for any satisfiable formula. For any sub-formula, *all* propositionally consistent sets are traversed. The number of propositionally consistent sets of sub-formulas is exponential in the length of the formula; therefore, with an explicit representation of sets these algorithms are limited to "small" formulas.

Natural models for linear time logics are sequences of points. Each point determines a propositionally consistent set of sub-formulas, namely the set of those sub-formulas which are valid in this point. Often, the number of different propositionally consistent sets determined by a specific linear-time model is small, compared to the number of all propositionally consistent sets. Thus, in the decision procedure it can be more appropriate to build a model incrementally:

- Start with some initial point, and
- iteratively choose the next point for the constructed sequence.

In this way, only those propositionally consistent sets have to be stored which actually appear in the model. Of course, in the worst case all propositionally consistent sets will be traversed; however, we can expect a better average-case behavior.

This procedure involves a nondeterministic choice. Therefore, it is implemented using backtracking search. Similar to the presentation in the previous subsection, we first give an algorithm for modal logic before considering operators which involve recursion. Since we are aiming at natural models, we use deterministic monomodal logic, that is, modal logic with a single accessibility relation R for which axiom **U** is required (cf. page 1691).

We want to decide whether a formula φ is satisfiable in a natural model globally validating the assumptions Φ. We start with the set W of all propositionally consistent extensions of $\Phi \cup \{\varphi\}$. That is, $w \in W$ iff

- $\varphi \in w$,
- $\Phi \subseteq w$,
- $\bot \notin w$, and
- for all sub-formulas $\psi = (\psi_1 \to \psi_2)$ it holds that $\psi \in w$ iff $\psi_1 \in w$ implies $\psi_2 \in w$.

We choose some $w_0 \in W$ and try to construct a model with w_0 as initial point. At level i in the construction, we are given a propositionally consistent set w_i. If it does not contain any formula $\langle R \rangle \psi$, we are finished. In this case, we have found a finite model of length i with final point w_i. Otherwise, we construct the set

$$w_i^R \triangleq \{\psi \mid \langle R \rangle \psi \in w_i\} \cup \{\neg\psi \mid \neg \langle R \rangle \psi \in w_i\} \cup \Phi$$

We refer to $\{\psi \mid \langle R \rangle \psi \in w_i\}$ as the *positive future obligations* and to $\{\neg\psi \mid \neg \langle R \rangle \psi \in w_i\}$ as the *negative future obligations* of w_i. Thus, w_i^R is the set of all future obligations of w_i (with respect to R), plus the global assumptions. We then build the set S of all propositionally consistent extensions of w_i^R. Since there are only finitely many subformulas of $(\hat{\Phi} \wedge \varphi)$, the set S is finite. If w_i^R is not propositionally consistent, then $S = \{\}$. In this case, we backtrack to level $i-1$ (or report failure, if $i = 0$). Otherwise, we choose some $w_{i+1} \in S$ as successor of w_i and continue *ad infinitum*. If we hit upon a point which is already contained in the constructed submodel $w_0 ... w_i$, then the infinite cyclic model $w_0 ... w_i (w_{i+1} ... w_i)^\omega$ initially satisfies φ and globally satisfies Φ. Since there are only finitely many maximal propositionally consistent sets, the construction must terminate. A pseudocode description of this algorithm is given in Figure 14.

procedure `ML_sat_lin` (Formula φ, Formulaset Φ) =
  $W := \{w \subseteq SF \mid \varphi \in w, \Phi \subseteq w, w$ propositionally consistent $\}$;
  $Stack := \{\}$;
  for all $w \in W$ **do** `depth_first_search`(w);
  `print`(φ, "is unsatisfiable with assumptions", Φ);

procedure `depth_first_search` (w) =
  if $w \in Stack$ **then** `print`(φ, Φ, "satisfiable by", *Stack*); **exit**;
  `push`(w, *Stack*);
  $pos := \{\psi \in SF \mid \langle R \rangle \psi \in w\}$; $neg := \{\psi \in SF \mid \langle R \rangle \psi \notin w\}$;
  if $pos = \{\}$ **then** `print`(φ, Φ, "satisfiable by", *Stack*); **exit**;
  $S := \{w \subseteq SF \mid pos \subseteq w, w \cap neg = \{\}, \Phi \subseteq w, w$ prop. consistent $\}$;
  for all $w' \in S$ **do** `depth_first_search`(w');
  `pop`(*Stack*);

Figure 14: Modal logic decision algorithm for linear models

In this algorithm, there is some redundant calculation.

- Firstly, whenever we backtrack from a point, there cannot be a successful continuation from this point. Therefore, we can add all points which are popped from the stack to a list M. If procedure depth_first_search is called with an argument which is contained in M, it can backtrack immediately.
- Secondly, we already noted that it is not necessary to represent a propositionally consistent set by an enumeration of all sub-formulas it consists of. It is sufficient to mark for every proposition variable and for every sub-formula starting with an $\langle R \rangle$-operator whether they are contained in the maximal consistent set.
- Thirdly, in the calculation of the set S of possible successors of a point, it is sufficient to consider propositionally consistent sets which are subsets of ($\boldsymbol{pos} \cup$ $\boldsymbol{neg} \cup \Phi$). That is, for sub-formulas ψ of φ which are neither future obligations of w nor sub-formulas of global assumptions from Φ, it is not necessary to fix their value in the successor w'. Both possible extensions, where $\psi \in w'$ or $\psi \notin w'$, are propositionally consistent and will lead to the same result. This improvement can be implemented for example by using a three-valued characteristic function for sub-formulas and propositionally consistent sets (contained, not contained, don't care) in the representation of points.

In Figure 15 we give a set of tableau rules for monomodal logic on deterministic models. The tableau rules can be seen as an implicit formulation of the algorithm in Figure 14, where the above improvements are included by definition. Similar tableau rules for modal logics can be found in [Fitting 1983] and for temporal logics in [Wolper 1985].

$$(\rightarrow)\ \frac{\Gamma, (\psi_1 \rightarrow \psi_2)}{\Gamma, \neg\psi_1 \qquad \Gamma, \psi_2} \qquad\qquad (\neg\rightarrow)\ \frac{\Gamma, \neg(\psi_1 \rightarrow \psi_2)}{\Gamma, \psi_1, \neg\psi_2}$$

$$(\bot_1)\ \frac{\Gamma, \psi, \neg\psi}{*} \qquad\qquad (\bot_2)\ \frac{\Gamma, \bot}{*} \qquad\qquad (\top)\ \frac{\Gamma, \neg\bot}{\Gamma}$$

$$(\langle R \rangle)\ \frac{\Gamma, \langle R \rangle \varphi_1, ..., \langle R \rangle \varphi_n, \neg \langle R \rangle \psi_1, ..., \neg \langle R \rangle \psi_m}{\Phi, \varphi_1, ..., \varphi_n, \neg\psi_1, ..., \neg\psi_m} \qquad\qquad ([R])\ \frac{\Gamma}{}$$

Figure 15: Tableau rules for monomodal logic on deterministic models

In these rules, Γ denotes any set of formulas, and Φ is the set of global assumptions. The double line in rules ($\langle R \rangle$) and ($[R]$) indicates a transition from one point in the constructed model to the next, and the star indicates that a branch is closed. Each tableau rule allows to derive zero, one or two sets of formulas from any set of formulas. Additional regulations are:

- Rule ($\rightarrow$) can only be applied if $\psi_2 \neq \bot$.
- Rules ($\langle R \rangle$) and ($[R]$) can only be applied if no other rule is applicable.
- Rule ($\langle R \rangle$) can only be applied if no other $\langle R \rangle \varphi$ or $\neg \langle R \rangle \psi$ is in Γ.

- Rule ($[R]$) can only be applied if no $\langle R\rangle\,\varphi$ is in Γ.

A *tableau* is a finite tree of sets of formulas such that

- The root of the tableau is $\Phi \cup \{\varphi\}$, and
- The children of each node are constructed according to some tableau rule.

A leaf is called *closed*, if it consists of the symbol $*$. It is called *open*, if it consists of a subset of formulas of some other node on the path from the root to this leaf. (In particular, if rule ($\langle R\rangle$) regenerates the root, the new leaf is open. Also, any empty node constructed by rule ($[R]$) is open). A tableau is *completed*, if any leaf is closed or open. A completed tableau is *successful*, if it contains an open leaf.

There is a strong connection between the tableau method and the local satisfiability algorithm sketched above. The propositional tableau rules systematically generate all necessary propositionally maximal consistent extensions of a given set of formulas, and the modal rules fix the structure of the accessibility relation(s) in the generated model graph.

For any given root, there are several different tableaus, since we did not specify any order in which the rules have to be applied. Nevertheless all these tableaus are equivalent: if there is *some* successful tableau for φ and Φ, then *every* completed tableau for it is successful.

7.2. Theorem. *φ is satisfiable with assumptions Φ iff $\Phi \cup \{\varphi\}$ has a successful tableau.*

Proof: For one direction, assume that there is some natural model $\mathcal{M} \triangleq ((w_0, w_1, w_2, ...), \mathcal{I}, w_0)$, where $w_0 \models \varphi$ and $w_i \models \Phi$ for every $i > 0$, and show that there is a completed tableau for φ and Φ with an open leaf. Equivalently, assume that any completed tableau for φ and Φ is given, and show that it contains an open leaf. We construct a sequence of tableau nodes n_i, and associate with any n_i a point $w(n_i)$ in the model. As an invariant of this construction, we show that for all formulas $\psi \in n_i$ it holds that $w(n_i) \models \psi$.

Initially n_0 is the root of the tableau, with $w(n_0) \triangleq w_0$. Since $w_0 \models \varphi$ and $w_0 \models \Phi$, the invariant is satisfied. Given any tableau node n_i with $w(n_i) = w_j$, no closing rules can be applicable, because this would contradict the invariant. Assume the child n_{i+1} of n_i is constructed by rule ($\neg \rightarrow$) or ($\top$). Then $w(n_{i+1}) \triangleq w_j$, and the invariant is preserved. If two children of n_i are constructed by rule ($\rightarrow$), then any one of them is chosen which preserves the invariant, and again $w(n_{i+1}) \triangleq w_j$. If n_i has a child obtained by rule ($\langle R\rangle$), then $w(n_{i+1}) \triangleq w_{j+1}$. The specific formulation of the rule guarantees that the invariant is preserved. Since the tableau is finite, and we can never apply one of the closing rules, we must hit an open leaf sooner or later.

For the other direction, we have to show that from any completed tableau with open leafs we can construct a model. The construction is similar to above. We consider the *unfolding* of the tableau. This is the tree arising from the repeated substitution of any open leaf with the subtableau rooted at the node subsuming this open leaf. In particular, an empty node generated by rule ($[R]$) can be replaced with any node on the path from the root to this node. If the tableau contains open leaves, then the unfolding contains infinite paths. In the unfolding, call any node whose child is constructed by rule ($\langle R\rangle$) or ($[R]$) a *pre-state*. It can be shown that the sequence of pre-states of any infinite path from the root constitutes an infinite model. $\square$

As an example for the tableau construction, consider the semaphore from page 1694. A linear time modelling of the transitions is given by

$$\Phi \triangleq \{(\mathtt{on} \to \mathtt{r} \ \ \wedge \langle R\rangle \neg\mathtt{on}), \quad (\neg\mathtt{on} \to \mathtt{r} \ \ \wedge \langle R\rangle \neg\mathtt{on} \vee \mathtt{s} \ \ \wedge \langle R\rangle\, \mathtt{on}), \quad \neg(\mathtt{r} \wedge \mathtt{s})\}.$$

Here, $\mathtt{r}$ and $\mathtt{s}$ denote the semaphore-operations "reset" and "set", respectively. We prove that after a reset the semaphore can be set only once:

$$\varphi \triangleq (\mathtt{r} \to [R](\mathtt{s} \to [R](\mathtt{s} \to \bot))).$$

To show that $\Phi \mid\!\vdash \varphi$, we construct the tableau with root $\Gamma \triangleq \Phi \cup \{\neg\varphi\}$ and show that it is closed. The tableau is given in Figure 16.

In this tableau, we omitted boolean decompositions. All leaves are closed because they contain both $\mathtt{r}$ and $\mathtt{s}$. This is a contradiction to the assumption $\neg(\mathtt{r} \wedge \mathtt{s})$ expressing that only one action is performed at a time.

Tableaus for **LTL**

We now extend these methods to linear time temporal logic. For simplicity, we restrict attention to the operators $\mathbf{X}$ and $\mathbf{F}^*$. The algorithms are similar to the modal logic case described above. To decide whether a formula φ is satisfiable in a natural model, we apply the same depth-first search algorithm as sketched in Figure 14, where $\mathbf{X}$ replaces $\langle R\rangle$, and extended sub-formulas (ESF) are used instead of subformulas (SF). (Recall that both ψ and $\mathbf{X}\,\mathbf{F}^*\psi$ are extended subformulas of $\mathbf{F}^*\psi$.)

There are two further modifications. Firstly, assume we are given a sub-formula $\mathbf{F}^*\psi$ and a node w such that one of the following holds.

- $\mathbf{F}^*\psi \in w$ and $\psi \notin w$ and $\mathbf{X}\,\mathbf{F}^*\psi \notin w$, or
- $\mathbf{F}^*\psi \notin w$, and $\psi \in w$ or $\mathbf{X}\,\mathbf{F}^*\psi \in w$.

In this case, we can discard node w. Even though it may be *propositionally* consistent, it does not respect the recursion axiom

$$\vdash \quad \mathbf{F}^*\psi \leftrightarrow \psi \vee \mathbf{X}\,\mathbf{F}^*\psi.$$

Thus, this node cannot appear as a point in the model.

Secondly, when the depth-first-search finds a node w which is already on the stack, it would be preliminary to report a success. Consider the set of nodes $w \triangleq w_0, w_1, ..., w_n$, which are on the path from w to w. It could be the case that there is some sub-formula $\mathbf{F}^*\psi$, such that each node contains both $\mathbf{F}^*\psi$ and $\mathbf{X}\,\mathbf{F}^*\psi$, but none of them contains ψ. That is, the eventuality φ is *required* but not *fulfilled* in

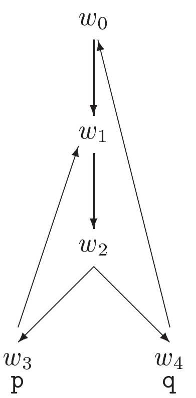
$$\Phi,\ \mathtt{r},\ \varphi_1 \qquad (\text{where } \varphi_1 \triangleq \langle R\rangle(\mathtt{s}\ \wedge \langle R\rangle\, \mathtt{s}))$$

$$\dfrac{\dfrac{\dfrac{\mathtt{on},\ \mathtt{r},\ \langle R\rangle \neg\mathtt{on},\ \varphi_1}{\Phi,\ \neg\mathtt{on},\ \mathtt{s},\ \langle R\rangle\, \mathtt{s}}}{\dfrac{\mathtt{r}, \mathtt{s}, \ldots}{*} \qquad \dfrac{\dfrac{\neg\mathtt{on},\ \mathtt{s}, \langle R\rangle\, \mathtt{s}, \langle R\rangle\, \mathtt{on}}{\Phi, \mathtt{s}, \mathtt{on}}}{\dfrac{\mathtt{s}, \mathtt{on}, \mathtt{r}, \langle R\rangle \neg\mathtt{on}}{*}}} \qquad \dfrac{\dfrac{\neg\mathtt{on},\ \mathtt{r},\ \langle R\rangle \neg\mathtt{on},\ \varphi_1}{\Phi,\ \neg\mathtt{on},\ \mathtt{s},\ \langle R\rangle\, \mathtt{s}}}{\ldots} \qquad \dfrac{\neg\mathtt{on},\ \mathtt{s},\ \langle R\rangle\, \mathtt{on},\ \mathtt{r},\ \varphi_1}{*}}{}$$

Figure 16: Tableau for Φ and $(\mathtt{r} \rightarrow [R](\mathtt{s} \rightarrow [R](\mathtt{s} \rightarrow \bot)))$.

$w_0, ..., w_n$. The fulfillment of $\mathbf{F}^* \varphi$ is "postponed forever" from each w_i to the next in a cyclic manner.

However, we cannot discard $w_0, w_1, ..., w_n$, because they might contribute to a satisfying model. Consider the following situation:



We assume that all nodes in this picture contain $\mathbf{F}^* \mathtt{p}$ and $\mathbf{F}^* \mathtt{q}$. The only node containing $\mathtt{p}$ is w_3, and the only node containing $\mathtt{q}$ is w_4. When the backtracking search encounters w_1 as child of w_3, it finds that in the loop w_1, w_2, w_3 the proposition $\mathtt{q}$ is required but not fulfilled. Thus it backtracks to w_2 and finds w_4 as second child. This time, in the loop w_0, w_1, w_2, w_4 the proposition $\mathtt{p}$ is required but not fulfilled. However, both $\mathtt{p}$ and $\mathtt{q}$ are satisfied in the model $(w_0, w_1, w_2, w_3, w_1, w_2, w_4)^\omega$.

Thus, when the depth-first-search finds a backward arc, it has to search the strongly connected component of nodes of the current node. A *strongly connected component* (SCC) is a set W of points such that for all $w_1, w_2 \in W$, if $w_1 \neq w_2$, then there is a path from w_1 to w_2 and back. An SCC W is called *terminal*, if for all $w \in W$, $w' \notin W$, it is not the case that $w \prec w'$. It is called *self-fulfilling*, if all required formulas are fulfilled, i.e., if for any $w \in W$ and $(\varphi_1 \mathbf{U}^+ \varphi_2) \in w$ there exists a $w' \in W$ such that $\varphi_2 \in w'$. For the decision algorithm, the depth-first-search graph has to be partitioned into strongly connected components. The given formula is satisfiable iff a self-fulfilling SCC is reachable from some initial node.

We postpone the algorithmic formulation of this partitioning to the next section, where the same algorithm is given in the context of model checking. Instead, we sketch the necessary modifications in the tableau construction. As additional rules to those of Figure 15, we add:

$$(\mathbf{F}^*)\ \frac{\Gamma, \mathbf{F}^* \psi}{\Gamma, \psi \qquad \Gamma, \mathbf{X}\,\mathbf{F}^* \psi} \qquad\qquad (\neg\,\mathbf{F}^*)\ \frac{\Gamma, \neg\,\mathbf{F}^* \psi}{\Gamma, \neg\psi, \neg\,\mathbf{X}\,\mathbf{F}^* \psi}$$

To deal with unfulfilled eventuality formulas, a backward loop can only be regarded as *open*, if for any $\mathbf{F}^* \psi$ which occurs in any w_i in the loop, there is a w_j in the loop such that $\psi \in w_j$ (*loop condition*). If the loop condition is not met, the unfolding of the tableau has to continue until all nodes of the SCC are contained in the loop. In this case, the respective branches are closed.

As an example for the loop condition, we show transitivity of $\leq$:

$$\vdash (\mathbf{F}^* \mathbf{F}^* \mathrm{p} \rightarrow \mathbf{F}^* \mathrm{p})$$

The root of the tableau is marked with the negation of this formula, i.e. with $\mathbf{F}^* \mathbf{F}^* \mathrm{p}$ and $\neg \mathbf{F}^* \mathrm{p}$.

$$\dfrac{\mathbf{F}^* \mathbf{F}^* \mathrm{p},\quad \neg \mathbf{F}^* \mathrm{p}}{\dfrac{\mathbf{F}^* \mathrm{p},\quad \neg \mathbf{F}^* \mathrm{p}}{*} \qquad\qquad \dfrac{\mathbf{X}\,\mathbf{F}^* \mathbf{F}^* \mathrm{p},\quad \neg \mathbf{F}^* \mathrm{p}}{\begin{array}{c} \mathbf{X}\,\mathbf{F}^* \mathbf{F}^* \mathrm{p},\quad \neg \mathrm{p},\quad \neg \mathbf{X}\,\mathbf{F}^* \mathrm{p} \\ \hline\hline \mathbf{F}^* \mathbf{F}^* \mathrm{p},\quad \neg \mathbf{F}^* \mathrm{p} \\ \hline ** \end{array}}}$$

The leaf marked $(*)$ closes because it is contradictory. The leaf $(**)$ closes because it is subsumed by the root above, and the SCC to which it belongs contains the unfulfilled eventuality $\mathbf{F}^* \mathbf{F}^* \mathrm{p}$.

There is a close connection between tableaus for temporal logics and ω-automata. The pre-states in the tableau (i.e., nodes immediately above a double line) can be seen as states of a generalized Büchi-automaton. The set of open leafs are the accepting states, and the recurring states are determined as follows: for any sub-formula $\varphi \triangleq \mathbf{F}^* \psi$ it must hold that φ is infinitely often not contained in an accepting run, or ψ is contained infinitely often. This can be formulated as generalized Büchi-acceptance condition on the states [Clarke, Grumberg and Hamaguchi 1997]. The formula then is satisfiable iff the language of the corresponding automaton is nonempty, and it is valid iff this language is Σ^ω (the set of all finite and infinite strings over Σ). Therefore, the decision problem for **LTL** can be regarded as an instance of the language problem of generalized Büchi automata [Wolper 1985, Emerson 1985, Vardi and Wolper 1986, Kurshan 1994]. In [Vardi 1995] other embeddings of tableau-based satisfiability procedures for temporal logics into decision algorithms for ω-automata, based on *alternating automata*, are described.

8. Basic Model Checking Algorithms

In this section, we will show how the most commonly used model checking procedures can be obtained from the above decision procedures.

Given a model $\mathcal{M}$ and a formula φ, the model checking problem is to decide whether $\mathcal{M} \models \varphi$. In principle, this can be done by encoding $\mathcal{M}$ as a set of assumptions ("premisses" or "program axioms") Φ, and deciding whether $\Phi \vdash \varphi$. However, some experiments will quickly convince the reader that a naïve approach of doing so is doomed to failure. Usually, the program axioms all have a very special form, such as

$(\texttt{state\_i} \to (\mathbf{X}\,\texttt{succ\_i1} \vee \cdots \vee \mathbf{X}\,\texttt{succ\_in}))$

in a linear time modelling, or

$(\texttt{state\_i} \to (\langle a_1\rangle\,\texttt{succ\_i1} \wedge \cdots \wedge \langle a_n\rangle\,\texttt{succ\_in}))$

in a branching time approach. The decision procedure in general can not take advantage of this special form of the assumptions and will in every step break down all assumptions to its basic propositional components. This results in a very inefficient behavior; usually only very small systems can be verified and debugged that way.

Therefore, model checking algorithms avoid the encoding of the models as a set of program axioms; they use the models directly instead. Model checking determines whether a given specification formula is satisfied in a given Kripke-model, i.e., whether a tree or natural model satisfying the formula can be generated from it.

There are two variants of this task, depending on whether the initial or universal definition of satisfaction of a formula in a model is used. In the usual definition, a Kripke-model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$ is given, which consists of universe U, accessibility relation(s) defined by $\mathcal{I}$, and current point $w_0 \in U$, and we have to check whether the formula φ is satisfied: $(U, \mathcal{I}, w_0) \models \varphi$. In the universal definition, we are given a frame $\mathcal{F} \triangleq (U, \mathcal{I})$ consisting of universe and interpretation, and want to know whether the formula is satisfied in *all* models based on this frame: $(U, \mathcal{I}) \models \varphi$ iff for all $w_0 \in U$ it holds that $(U, \mathcal{I}, w_0) \models \varphi$. Equivalently, we want to know whether $\varphi^{\mathcal{F}} = U$, where $\varphi^{\mathcal{F}} \triangleq \{w \in U \mid w \models \varphi\}$ is the set of points satisfying φ.

Of course, any algorithm which calculates $\varphi^{\mathcal{F}}$ can also be used to decide whether $(U, \mathcal{I}, w_0) \models \varphi$ holds: $w_0 \models \varphi$ iff $w_0 \in \varphi^{\mathcal{F}}$. Vice versa, if we have an efficient algorithm to decide whether $w_0 \models \varphi$, we can calculate $\varphi^{\mathcal{F}}$ by an iteration on all states.

The model checking problem has two parameters: model $\mathcal{M}$ and formula φ. Algorithms which iterate on the structure of φ and in each step traverse the whole of $\mathcal{M}$ are sometimes called *global*. Algorithms which iteratively extend the checked part of $\mathcal{M}$ and in each step determine the truth of each sub-formula of φ are sometimes called *local*. Although the theoretical worst-time complexity is not influenced by this choice, the average case behavior may differ significantly.

In principle, the three axes (branching/linear, universal/initial, global/local) are independent. In practice, however, for branching time logics mostly global algorithms for universal validity are used, whereas for linear time logics local algorithms for initial validity have been suggested.

8.1. Global Branching Time Model Checking

Given a Kripke frame $\mathcal{F} = (U, \mathcal{I})$ and a multimodal formula φ, the set $\varphi^{\mathcal{F}} \triangleq \{w \in U \mid w \models \varphi\}$ of points validating φ can be calculated by a recursive descent on the structure of φ:

- If $\mathtt{p}$ is an atomic proposition, then $\mathtt{p}^{\mathcal{F}} \triangleq \mathcal{I}(\mathtt{p})$.
- $\bot^{\mathcal{F}} \triangleq \{\}$.
- $(\varphi \to \psi)^{\mathcal{F}} \triangleq (U \setminus \varphi^{\mathcal{F}}) \cup \psi^{\mathcal{F}}$.
- $(\langle R \rangle \psi)^{\mathcal{F}} \triangleq \{w \in U \mid \exists w' \in \psi^{\mathcal{F}}, (w, w') \in \mathcal{I}(R)\}$.

This algorithm seems to be just a trivial reformulation of the semantical definition for the logical operators. However, there are some important observations.

- Firstly, $(\langle R \rangle \psi)^{\mathcal{F}}$ can be calculated from $\psi^{\mathcal{F}}$ in two ways: we can check for each $w \in U$, whether the intersection of $\psi^{\mathcal{F}}$ and $R(w)$ is nonempty. Alternatively, we can calculate $\bigcup\{R^{-1}(w') \mid w' \in \psi^{\mathcal{F}}\}$, where $R^{-1}(w') \triangleq \{w \mid (w, w') \in \mathcal{I}(R)\}$ is the *inverse image* of point w under the relation R. This inverse image calculation can be accomplished by a traversal of all arcs $(w, w') \in \mathcal{I}(R)$: if $w' \in \psi^{\mathcal{F}}$, then $w \in (\langle R \rangle \psi)^{\mathcal{F}}$.
- Secondly, to avoid recalculation of common subformulas, we use a table, where for each sub-formula ψ the set $\psi^{\mathcal{F}}$ is stored. The size of $\psi^{\mathcal{F}}$ can be of the same order of magnitude as $|U|$. Thus, we need an efficient data structure for large sets of points.
- Thirdly, the overall *complexity* of this algorithm is linear in the number of different sub-formulas and in the size of the model. However, even for infinite models which are given by some symbolic description, e.g., Petri nets or Turing machines, some model checking problems can be decidable [Andersen 1994, Gurov, Berezin and Kapron 1996, Burkart and Esparza 1997]. In such cases, $\psi^{\mathcal{F}}$ can be of infinite size, and must be represented by a symbolic description as well.

Similar to the above modal logic procedure, the **CTL** model checking algorithm proceeds by marking each point with the set of sub-formulas which are valid for this point. Suppose we have already marked the set of points satisfying ψ_1 and the points satisfying ψ_2. To label the set of points satisfying $\varphi \triangleq \mathbf{E}(\psi_2\, \mathbf{U}^+\, \psi_1)$ or $\varphi \triangleq \mathbf{A}(\psi_2\, \mathbf{U}^+\, \psi_1)$, we use the fixpoint unfoldings

$$\mathbf{E}(\psi_2\, \mathbf{U}^+\, \psi_1) \leftrightarrow \mathbf{E}\,\mathbf{X}(\psi_1 \vee \psi_2 \wedge \mathbf{E}(\psi_2\, \mathbf{U}^+\, \psi_1))$$

$$\mathbf{A}(\psi_2\, \mathbf{U}^+\, \psi_1) \leftrightarrow \mathbf{A}\,\mathbf{X}(\psi_1 \vee \psi_2 \wedge \mathbf{A}(\psi_2\, \mathbf{U}^+\, \psi_1))$$

For $\varphi \triangleq \mathbf{E}(\psi_2\, \mathbf{U}^+\, \psi_1)$, we label all points with φ which have a successor that is labelled with ψ_1, or with ψ_2 and also φ. This process is repeated until stabilization is reached. For $\varphi \triangleq \mathbf{A}(\psi_2\, \mathbf{U}^+\, \psi_1)$, note that $\mathbf{A}\,\mathbf{X}\,\chi \leftrightarrow (\mathbf{E}\,\mathbf{X}\,\top \wedge \mathbf{A}\,\cancel{\mathbf{X}}\,\chi)$. Thus, we label all points with φ which have at least one successor, and for which all successors are labelled with ψ_1, or with ψ_2 and also φ. Again, this process must be repeated until no new points can be marked. The procedure is comparable to the marking algorithm in Figure 13. A recursive formulation of this algorithm is given in Fig. 17.

Since the Kripke-model has a finite number of points, each **repeat** in the algorithm stabilizes after at most $|U|$ passes. In the worst case, each pass searches the whole model ($|U|^2$ transitions), hence the complexity is linear in the number of different sub-formulas, and cubic in $|U|$.

procedure `CTL_check` (Model $(U, \mathcal{I}, w_0)$, Formula φ) =
    if $w_0 \in$ `eval`(φ)
    then `print`("φ is satisfied at w_0 in $(U, \mathcal{I})$")
    else `print`("φ not satisfied at w_0 in $(U, \mathcal{I})$");

function `eval` (Formula φ): Pointset =
    case φ **of**
        `p` : **return** $\mathcal{I}$(`p`);
        $\perp$: **return** $\{\}$;
        $(\psi_1 \to \psi_2)$: **return** $U \setminus$ `eval`$(\psi_1) \cup$ `eval`(ψ_2);
        $\mathbf{E}(\psi_2\, \mathbf{U}^{+}\, \psi_1)$: $E1 :=$ `eval`(ψ_1); $E2 :=$ `eval`(ψ_2); $E := \{\}$;
            repeat until stabilization
                $E := E \cup \{w \mid (\mathtt{succ}(w) \cap (E1 \cup (E2 \cap E))) \neq \{\}\}$;
            return E;
        $\mathbf{A}(\psi_2\, \mathbf{U}^{+}\, \psi_1)$: $E1 :=$ `eval`(ψ_1); $E2 :=$ `eval`(ψ_2); $E := \{\}$;
            repeat until stabilization
                $E := E \cup \{w \mid \{\} \neq \mathtt{succ}(w) \subseteq E1 \cup (E2 \cap E)\}$;
            return E;
function `succ` (Point w): Pointset = **return** $\{w' \mid (w, w') \in \mathcal{I}(\prec)\}$;

Figure 17: naïve **CTL** model checking algorithm

This bound can be improved if the search is organized better. In [Clarke, Emerson and Sistla 1986], an algorithm is given which is linear in the size of the model as well. For the $\mathbf{E}\,\mathbf{F}^{+}$-operator, the problem of marking all points for which $\mathbf{E}\,\mathbf{F}^{+}\,\varphi$ holds, given the set of point satisfying φ, is equivalent to the *inverse reachability problem*: given a set of points, mark all points from which any finite path leads into the given set. Assuming that for any two points we can decide in constant time whether they are connected by an arc, this can be done with time complexity quadratic in the number of points.

function `reach` (Pointset *Target*): Pointset =
    Source := $\{\}$; *Search* := *Target*;
    while *Search* $\neq \{\}$ **do**
        Search := `pred` (*Search*) $\setminus$ *Source*;
        Source := *Source* $\cup$ *Search*
    enddo;
    return *Source*;
function `pred` (Point w): Pointset = **return** $\{w' \mid (w', w) \in \mathcal{I}(\prec)\}$;

Figure 18: Inverse reachability calculation

The algorithm given in Fig. 18 calculates the set *Source* of all points from which any point in given set *Target* is reachable. In this algorithm, every point enters the set *Search* in the **while** loop at most once. Moreover, all set operations can be performed in time linear in the size of these sets, i.e., in the number of points; thus the overall complexity is quadratic in $|U|$ or linear in the size of the Kripke-model.

For the $\mathbf{E}\,\mathbf{U}^{+}$-operator, this idea can be refined to give an evaluation procedure of linear complexity. The $\mathbf{A}\,\mathbf{U}^{+}$-operator can be expressed by

$$\mathbf{A}(\psi_2\,\mathbf{U}^{+}\,\psi_1) \leftrightarrow \neg(\mathbf{E}(\neg\psi_1\,\mathbf{U}^{+}(\neg(\psi_1 \wedge \psi_2)) \vee \mathbf{E}\,\mathbf{G}^{+}\,\neg\psi_1)$$

Thus, we only need a procedure marking all points for which $\mathbf{E}\,\mathbf{G}^{+}\,\varphi$ holds. This can be done as follows:

- restrict the model to those states satisfying φ
- find the maximal strongly connected components in the restriction
- mark all points in the original model from which a nontrivial SCC or a point without successors can be reached by a path in the restricted model.

These operations can be accomplished with time complexity which is quadratic in U. Thus, the overall complexity of CTL model checking is linear in the size of the formula and in the size of the model.

Fairness Constraints

Some automated model checkers for **CTL** (for example, SMV [McMillan 1993] and SVE [Dingel and Filkorn 1995]) allow to specify a set of *constraints* Φ together with the Kripke-model. These constraints are assumed to hold in the whole model; i.e., they restrict the model to those parts where they are valid. This use of constraints is somewhat different from the assumptions in the previous sections, which were used to constrain the *set* of possible models. For example, an ω-automaton can be regarded as a Kripke-model, together with global eventuality and fairness constraints (accepting and recurring states). Constraints can be formulated in the same language in which the formula to be checked is specified; however, "mixed" approaches have been suggested [Josko 1993], where e.g. the constraints are described in **LTL** and the property is described in **CTL**.

As an example for the use of such constraints, often the path-quantifiers **A** and **E** are restricted to *fair* paths. *Simple* fairness constraints are of form $\mathbf{F}^{+}\,\psi$, where ψ is a boolean combination of propositions. For example, the condition $\mathbf{F}^{+}\,\top$ specifies that each run must be infinite. As another example for a simple fairness constraint, we might want to restrict our attention to execution sequences in which every component is always eventually scheduled. Streett fairness constraints are of form $(\mathbf{G}^{+}\,\mathbf{F}^{+}\,\psi_1 \rightarrow \mathbf{G}^{+}\,\mathbf{F}^{+}\,\psi_2)$ and are useful to restrict attention to *strongly fair* schedulers: if a component infinitely often requests a resource, it will be granted infinitely often. Historically, different fairness constraints were discussed in [Lehmann, Pnueli and Stavi 1981, Quielle and Sifakis 1982]. A comprehensive treatment of fairness concepts and proofs is given in [Francez 1986].

The above algorithm can be modified to deal with such fairness constraints by building the tableau of the **LTL**-assumption and checking the **CTL**-formula on the product of Kripke-model and tableau. The complexity increases by a factor which depends on the type of **LTL**-formulas in the assumption. For more information, see [Emerson and Lei 1986, Kupferman and Vardi 1996, Emerson, Jutla and Sistla 1993, Clarke, Grumberg and Long 1993].

8.2. Local Linear Time Model Checking

For a given Kripke-model $\mathcal{M} = (U, \mathcal{I}, w_0)$ and **CTL**-formula φ, the relation $\mathcal{M} \models \varphi$ holds iff the maximal tree generated from $\mathcal{M}$ at w_0 satisfies φ. For linear time logics, $\mathcal{M} \models \varphi$ is interpreted by *sequence-validity.* That is, we want to check whether *every maximal sequence* generated from $\mathcal{M}$ at w_0 satisfies φ. Equivalently, we have to decide whether $\neg\varphi$ is satisfiable in *some* natural model generated from $\mathcal{M}$. In some sense, this is a more complex question than the one for branching time, because a whole *set* of natural models has to be checked. Hence, we cannot simply mark a point in the Kripke-model with the set of linear-time formulas which are valid for this point: for example, $\mathbf{F}^{+}\,\psi$ can be valid for one of the generated sequences, and not valid for another one.

We first consider sequence-validity of modal logic with a single accessibility relation R. Given a Kripke-model $\mathcal{M} = (U, \mathcal{I}, w_0)$ and a modal formula φ, we want to determine whether there is a maximal sequence generated from $\mathcal{M}$ at w_0 which satisfies φ in w_0. This is done by a depth-first-search in the product of the set of propositionally maximal consistent sets of sub-formulas and the set of points in the model.

Formally, an *atom* α is any pair (w, m), where $w \in U$ is a point, and $m \subseteq SF(\varphi)$ is a propositionally consistent set of sub-formulas. An atom is *admissible*, if w and m agree on the interpretation of propositions. That is, if $\mathtt{p} \in SF(\varphi)$, then $\mathtt{p} \in m$ iff $w \in \mathcal{I}(\mathtt{p})$.

An *initial atom* is any admissible atom $\alpha = (w_0, m_0)$, where w_0 is the current point of $\mathcal{M}$, and $\varphi \in m_0$. We define a relation X_R between admissible atoms: $X_R((w, m), (w', m'))$ holds iff the following conditions are met:

1. $(w, w') \in \mathcal{I}(R)$,
2. if $\langle R\rangle\,\psi \in SF(\varphi)$ and $\psi \in m'$, then $\langle R\rangle\,\psi \in m$,
3. if $\langle R\rangle\,\psi \in m$, then $\psi \in m'$, and
4. some $\langle R\rangle\,\psi \in m$.

The first condition reflects the fact that the steps in the generated sequence are predetermined by the Kripke-model. The second condition is imposed by the semantics of the $\langle R\rangle$-operator. The third condition is a reformulations of the axiom **(U)** and the corresponding tableau rule $(\langle R\rangle)$ on page 1706. The fourth condition corresponds to the tableau rule $([R])$; it allows the generated sequence to be finite when no $\langle R\rangle\,\psi$ is contained in a node.

Now we can construct a forest of atoms as follows:

- initial nodes are all initial atoms

- any node α has as children all α' such that $X_R(\alpha, \alpha')$

Since for any finite Kripke-model there are only finitely many atoms, each branch in this forest can be made finite by appropriate backward arcs. As in the tableau definition, a leaf is called *open*, if it has no $\langle R \rangle$ formulas in its first component (m); otherwise, it is *closed*.

An *accepting path* through the resulting structure starts with any initial node and is either infinite or ends with an open leaf. Any accepting path is a sequence generated from the Kripke-model which satisfies the given formula $\neg\varphi$, thereby forming a counterexample to the specification φ.

To implement the search for an accepting path, we perform a depth-first search with backtracking from the set of initial atoms to all of its successors. In order to be able to terminate loops in this search, we have to store all atoms which were encountered previously. Though there are several possibilities to represent such a set of atoms, the method of choice seems to be to employ a hash table. It is not necessary to use all components of m as hash indices, since the value of propositions is determined by w, and boolean combinations of formulas can be recovered from their constituent parts. Therefore, it is sufficient to store only the value of $\langle R \rangle$-subformulas.

In general, since we are only looking for *some* counter-model, we can terminate the search if a counter-model is found. Although in the worst case (if no counter-model exists) the whole forest must be searched, it is possible to find errors very quickly by an appropriate ordering of the depth-first search successors.

In the depth-first search, we have to remove closed atoms from the list of possible loop points. A better way is to *mark* these nodes as closed while backtracking; then the search will not recurse again if such an atom reappears. Also all other improvements mentioned on page 1706 can be used for this algorithm.

Extensions for **LTL**

We have seen that the local model checking algorithm for modal logic is almost the same algorithm as the local tableau decision procedure. Similarly, the local model checking for **LTL** is very close to its respective satisfiability algorithm. For simplicity, in this subsection we restrict attention to the future fragment of **LTL**.

In the definition of $X_R((w, m), (w', m'))$, we replace $\langle R \rangle$ by $\mathbf{X}$ and require in addition

5. if $\mathbf{F}^* \psi \in SF(\varphi)$ then $\mathbf{F}^* \psi \in m$ iff $\psi \in m$ or $\mathbf{X}\,\mathbf{F}^* \psi \in m$

This requirement corresponds to the recursion axiom $\vdash \mathbf{F}^* \psi \leftrightarrow \psi \vee \mathbf{X}\,\mathbf{F}^* \psi$. As in the case of modal logic, we try to thread an accepting path through the graph of atoms which arises from this definition. However, we can only accept those paths in which all eventualities $\mathbf{F}^* \psi$ are fulfilled. Since we can not guarantee that several eventualities are simultaneously fulfilled in some single loop, we have to calculate the strongly connected components of the reflexive transitive closure of X_R. An SCC W of atoms is called *self-fulfilling*, if for any $\mathbf{F}^* \psi$ in some $\alpha \in W$ there exists some $\alpha' \in W$ with $\psi \in \alpha'$. Any atom which does not contain positive future obligations $\mathbf{X}\,\psi$ is a trivial SCC, because it is a terminal node in the atom graph. Such a node forms a self-fulfilling SCC, because the above condition (5.) guarantees that for any $\mathbf{F}^* \psi \in \alpha$, also $\psi \in \alpha$. The given formula φ is satisfiable in $\mathcal{M}$ iff there exists a self-fulfilling SCC which is reachable from some initial atom. In this case, a natural model for φ generated by $\mathcal{M}$ is given by any sequence of atoms from an initial atom which ends in a terminal atom or infinitely often passes through all atoms of a self-fulfilling SCC.

For $\mathbf{U}^+$-operators, each positive occurrence $(\psi_1 \, \mathbf{U}^+ \, \psi_2)$ in some $\alpha \in W$ is an eventuality which has to be fulfilled at some point; thus the SCC W is defined to be self-fulfilling, if it is nontrivial and for any $(\psi_1 \, \mathbf{U}^+ \, \psi_2)$ in some $\alpha \in W$ there exists some $\alpha' \in W$ with $\psi_2 \in \alpha'$, or it is trivial and does not contain any $(\psi_1 \, \mathbf{U}^+ \, \psi_2)$.

To construct maximal SCCs, two different algorithms have been suggested(see, e.g. [Aho, Hopcroft and Ullman 1974]). For model checking, Tarjan's algorithm [Tarjan 1972] is particularly well-suited, since it enumerates the strong components of a graph during the backtrack from the depth-first search. If a maximal SCC W is found, all required and fulfilled eventualities in all nodes of W can be collected. W is self-fulfilling if all required eventualities are fulfilled. Thus model checking can be performed "on-the-fly" during the enumeration of the reachable atoms of the model. An appropriate depth–first–search **LTL** model checking algorithm is given in 19.

In this algorithm, the function `children` constructs for a given atom α the set of all possible successor atoms according to the transition relation of the Kripke-model and to the fixed point definition of the until-operator. One way to implement this function is to represent atoms by bitstrings which contain one bit for each proposition $\mathtt{p} \in \mathcal{P}$ and one bit for each sub-formula $(\psi_2 \, \mathbf{U}^+ \, \psi_1) \in SF(\varphi)$. New atoms are included into a hash table, which contains one bitstring for each atom. For each entry into the hash table, the function `children` returns a list of pointers to the hash table. For more information on bitstate hashing techniques and state space caching, see [Courcoubetis, Vardi, Wolper and Yannakakis 1992, Holzmann 1995, Godefroid, Holzmann and Pirottin 1995].

The procedure `depth_first_search` realizes Tarjan's algorithm and the test whether an SCC is self-fulfilling. It recursively builds all atoms reachable from a given atom α. When the procedure backtracks, α is the root of a maximal SCC iff there are no atoms β in the subtree below α such that α is also in the subtree of β. In this case, the maximal SCC containing α consists of all nodes in the subtree below α, and this maximal SCC can be checked for acceptance. *table* is implemented as a hash table from atoms to natural numbers. *table*[α] contains

- UNDEFINED, as long as atom α has not occurred,
- the depth-first-number of α, when α is first encountered,
- the depth–first–number of the first encountered atom belonging to the same strongly connected component as α, after return from the recursive call, and
- MAXNAT (any value for which $\mathtt{min}(n, \text{MAXNAT})$ is always n), after the maximal strong component containing α has been analyzed.

To check whether an SCC is self-fulfilling, during its enumeration two sets are built: *required* contains the union of all eventualities which are required, and *fulfilled*

```
procedure LTL_check (Model 𝓜, Formula φ) =
    Nat depth_first_count := 0;    /* number of recursive call */
    Atomset stack := {};    /* Stack of searched atoms */
    Natarray table;    /* Hashtable from atoms to natural numbers */
    Atomset init := {α | α is an initial atom of 𝓜 and φ};
    for all α ∈ init do depth_first_search(α);
    print("φ is not satisfiable in 𝓜");

procedure depth_first_search (Atom α) =
    if (table [α] = UNDEFINED) then    /* α is a new atom */
        Nat dfnumber := depth_first_count;    /* save current count */
        depth_first_count := depth_first_count+1;
        table[α] := dfnumber;    /* initialize with current depth */
        push(stack, α);
        Atomset succ := children(α);
        for all (β ∈ succ) do
            depth_first_search(β);
            table[α] := min(table[α], table[β]);    /* β above α? */
        if (table[α] = dfnumber) then    /* α is the root of an SCC */
            Formulaset required := {}, fulfilled := {};
            repeat
                β := pop(stack);
                table[β] := MAXNAT;
                required := required ∪ {ψ1 | (ψ2 U⁺ ψ1) ∈ β};
                fulfilled := fulfilled ∪ {ψ | ψ ∈ β}
            until (α = β);    /* all elements of SCC are popped */
            if required ⊆ fulfilled    /* SCC is self-fulfilling */
            then print("φ satisfiable in 𝓜"); exit;

function children (Atom (w, m)) : Atomset =
    if {(ψ2 U⁺ ψ1) ∈ m} = {} then return {}    /*no future obligations*/
    else return {(w′, m′) | w ≺ w′,
                (ψ2 U⁺ ψ1) ∈ m iff ψ1 ∈ m′ or ψ2 ∈ m′ and (ψ2 U⁺ ψ2) ∈ m′}
```

Figure 19: Depth–first–search **LTL** model checking algorithm

contains the union of all eventualities which are fulfilled in the atoms of this SCC. The SCC is self-fulfilling if *required*$\subseteq$*fulfilled.*

The main program calls `depth_first_search` for all initial atoms, where for an initial atom (w_0, m_0)

1. w_0 is the current point of $\mathcal{M}$, and
2. $m_0 \subseteq SF(\varphi)$ is any propositionally consistent set such that $\varphi \in m_0$.

If during the construction of the atom graph a maximal self-fulfilling SCC is found, the algorithm reports success; if the whole graph is searched without success we know that the formula is not satisfiable, and the program terminates with this result.

This algorithm is exponential in the number of $\mathbf{U}^+$-formulas, because every set of such sub-formulas determines a propositionally consistent set. It is linear in the size of the Kripke-model. In general, it can be shown that the problem of **LTL**-model checking (including past-operators) is PSPACE-complete in the size of the formula and NLOGSPACE in the size of the model (see [Sistla and Clarke 1986, Lichtenstein and Pnueli 1985]). The exponential complexity in the length of the formula usually is not very problematic, because specification formulas tend to be rather short. The linear complexity in the size of the model is a more serious limiting factor, since in the worst case (i.e., if the formula is unsatisfiable) all atoms have to be traversed. Current technology limits the applicability of such algorithms to models with approximately $10^5 - 10^6$ reachable atoms. In Section 11 we will discuss approaches which try to overcome this limit.

8.3. Model Checking for Propositional μ-Calculus

Both the local and the global model checking algorithms can be easily adapted to μ**TL**. Global model checking for **CTL** unfolds the fixpoint definition of the $\mathbf{A}\,\mathbf{U}^+$ and $\mathbf{E}\,\mathbf{U}^+$ operators. If we restrict our attention to *continuous* μ**TL**-formulas (see below), then this idea can be used to obtain a global model checking algorithm for these formulas. Moreover, as we will discuss in Section 10, this algorithm can be efficiently implemented using BDDs (see [Burch, Clarke, McMillan, Dill and Hwang 1992]).

According to the Knaster-Tarski theorem proved in Section 3.2,

$$(U, \mathcal{I}, w) \models \nu q\ \varphi \ \text{ iff } \ w \in \bigcup\{Q \mid Q \subseteq \varphi^{\mathcal{F}}\{q := Q\}\}$$

$$(U, \mathcal{I}, w) \models \mu q\ \varphi \ \text{ iff } \ w \in \bigcap\{Q \mid \varphi^{\mathcal{F}}\{q := Q\} \subseteq Q\}$$

A function $f : 2^U \to 2^U$ is called *union-continuous*, if $f(\bigcup_{i \in I}\{x_i\}) = \bigcup_{i \in I} f(x_i)$ for any index set I. If the functional defined by φ is union-continuous, then the fixpoints can be obtained as

$$\nu q\ \varphi = lim_{i \to \omega} \varphi^i(\top)$$

$$\mu q\ \varphi = lim_{i \to \omega} \varphi^i(\bot)$$

If U is finite, then every monotonic function is union-continuous. Moreover, according to Lemma 5.4, on finite models it is sufficient to consider the limit up to the cardinality of the universe:

$$\nu q\ \varphi = lim_{i \leq |U|} \varphi^i(\top)$$

$$\mu q\ \varphi = lim_{i \leq |U|} \varphi^i(\bot)$$

```
function eval (Formula φ): Pointset =
    case φ of
        p : return I(p); /* interpretation of proposition p */
        q : return v(q); /* valuation of proposition variable q */
        ⊥ : return {};
        (ψ1 → ψ2) : return U\ eval(ψ1) ∪ eval(ψ2);
        ⟨R⟩ ψ : return R^-1( eval(ψ));
        νq(ψ) : H := U;
            repeat until stabilization
                H := eval(ψ{q := H});
            return H;
        μq(ψ) : H := {};
            repeat until stabilization
                H := eval(ψ{q := H});
            return H;
```

Figure 20: naïve global branching time $\mu\mathbf{TL}$ model checking algorithm

Consequently, for finite domains model checking of positive $\mu\mathbf{TL}$ can be performed by extending the naïve global algorithm. The result is depicted in Figure 20.

Since every **repeat** in this algorithm can iterate up to $|U|$ times, the complexity is of order $|\varphi| \cdot |U|^{qd(\varphi)}$, where $qd(\varphi)$ is the depth of nesting of fixpoint operators in φ. This high complexity is due to the fact that the computation of any inner fixed point formula has to be restarted from scratch for every new iteration of an enclosing fixed point operator. For example, consider the **CTL**-formula $\mathbf{E}\,\mathbf{F}^{*}(\mathtt{p}_1 \wedge \mathbf{E}\,\mathbf{F}^{*}\,\mathtt{p}_2)$.

$$\mu\mathbf{TL}(\mathbf{E}\,\mathbf{F}^{*}(\mathtt{p}_1 \wedge \mathbf{E}\,\mathbf{F}^{*}\,\mathtt{p}_2)) = \mu q_1(\mathbf{X}\,q_1 \vee (\mathtt{p}_1 \wedge \mu q_2(\mathbf{X}\,q_2 \vee \mathtt{p}_2))).$$

This formula is *alternation-free*: in the inner fixed point formula $\mu q_2(\mathbf{X}\,q_2 \vee \mathtt{p}_2)$ there is no occurrence of q_1. Therefore, in the evaluation of μq_1, this formula has a constant value. For such formulas, model checking can be done with linear time complexity [Emerson et al. 1993, Cleaveland and Steffen 1993]. In contrast, consider the $\mu\mathbf{TL}$ formula

$$\mu q_1(\mathtt{p}_1 \wedge \mu q_2(\mathbf{X}\,q_1 \vee \mathbf{X}\,q_2 \vee \mathtt{p}_2)).$$

Here the inner formula $\mu q_2(\mathbf{X}\,q_1 \vee \mathbf{X}\,q_2 \vee \mathtt{p}_2)$ is re-evaluated for every new iteration of q_1. That is, if $\psi(q_1, q_2) \triangleq (\mathbf{X}\,q_1 \vee \mathbf{X}\,q_2 \vee \mathtt{p}_2)^{\mathcal{F}}$ and $\varphi(q_1) \triangleq (\mathtt{p}_1 \wedge \mu q_2 \psi(q_1, q_2))^{\mathcal{F}}$, we calculate $\mu q_1 \varphi(q_1)$ by iterating

$\varphi^0 \triangleq \bot$,

$\psi^{0,0} \triangleq \bot$

$\psi^{0,1} \triangleq \psi(\varphi^0, \psi^{0,0}) = (\mathbf{X} \bot \vee \mathbf{X} \bot \vee \mathtt{p}_2)$,

$\psi^{0,2} \triangleq \psi(\varphi^0, \psi^{0,1}) = (\mathbf{X} \bot \vee \mathbf{X}\,(\mathbf{X} \bot \vee \mathtt{p}_2) \vee \mathtt{p}_2)$,

$$
\begin{aligned}
&\quad \psi^{0,3} \triangleq \psi(\varphi^0, \psi^{0,2}) = (\mathbf{X}\perp \vee \mathbf{X}\,\big(\,\mathbf{X}\perp \vee \mathbf{X}(\mathbf{X}\perp \vee \mathtt{p}_2) \vee \mathtt{p}_2\big) \vee \mathtt{p}_2),\\
&\quad \ldots\\
&\quad \psi^{0,n+1} \triangleq \psi(\varphi^0, \psi^{0,n}) = \mu q_2(\mathbf{X}\perp \vee \mathbf{X}\,q_2 \vee \mathtt{p}_2),\ \text{ if } \psi^{0,n+1} = \psi^{0,n},\\
&\varphi^1 \triangleq \varphi(\varphi^0) = (\mathtt{p}_1 \wedge \mu q_2(\mathbf{X}\perp \vee \mathbf{X}\,q_2 \vee \mathtt{p}_2)) = (\mathtt{p}_1 \wedge \psi^{0,n}),\\
&\quad \psi^{1,0} \triangleq \perp\\
&\quad \psi^{1,1} \triangleq \psi(\varphi^1, \psi^{1,0}) = (\mathbf{X}\,\big(\mathtt{p}_1 \wedge \mu q_2(\mathbf{X}\perp \vee \mathbf{X}\,q_2 \vee \mathtt{p}_2)\big) \vee \mathbf{X}\perp \vee \mathtt{p}_2),\\
&\quad \psi^{1,2} \triangleq \psi(\varphi^1, \psi^{1,1}) = (\mathbf{X}\,\varphi^1 \vee \mathbf{X}\,\psi^{1,1} \vee \mathtt{p}_2),\\
&\ldots
\end{aligned}
$$

and so on. A more sophisticated algorithm was given in [Emerson and Lei 1986]. Each sequence $\nu q_1 ... \nu q_n$ or $\mu q_1 ... \mu q_n$ of nested fixpoints of the same type can be calculated by a single loop. Since ψ is monotonic, and $\varphi^0 \subseteq \varphi^1$, we have $\psi^{0,n} \subseteq \psi^{1,n}$. To compute a least fixed point, it is sufficient to start with any value below the result. Therefore, $\psi^{1,0}$ can be initialized with $\psi^{0,n}$ instead of $\perp$. Generally, when restarting the computation of an inner fixed point of the same type, we can use the last approximation result as a starting value. Thus, the value of this inner fixed point can increase at most $|U|$ times. The overall complexity of this improved algorithm is $(|\varphi| \cdot |U|)^{ad(\varphi)}$, where $ad(\varphi)$ is the alternation depth of different fixpoint operators in φ.

In [Long, Browne, Clarke, Jha and Marrero 1994] the authors observe that by storing even more intermediate values, the time complexity for evaluating fixpoint formulas can be reduced to $O(|U|^{\lfloor ad/2 \rfloor + 1})$. It can be shown that the complexity of model checking $\mu\mathbf{TL}$ is in NP $\cap$ co-NP; however, no lower bound is known to date. For more information, see [Berezin, Clarke, Jha and Marrero 1996].

For the local version, there have been a number of algorithms proposed in the literature [Winskel 1991, Cleaveland 1990, Bradfield and Stirling 1991, Stirling 1991]. We give a sketch of the tableau method from [Stirling and Walker 1991], which illustrates the basic ideas. The algorithm explores only a (small) part of the model by depth-first search. Each node in the tableau is marked by a sequence $\Delta, w \models \psi$, where $w \in U$ is a point in the model, ψ is a sub-formula of the given formula and Δ is a *definition list*. This is a sequence of declarations $(q_1 = \psi_1, ..., q_n = \psi_n)$, where the proposition variables q_i are pairwise disjoint and ψ_i uses at most variables from $q_1, ..., q_{i-1}$. For simplicity, we use $\vee$, $\wedge$, $\langle R \rangle$, $[R]$, μ and ν as basic operators and assume that negations only occur in literals. Furthermore, we assume that in the formula to be checked each μ and ν quantification binds a different proposition variable.

Since in [Stirling and Walker 1991] the μ-calculus is interpreted on branching structures, the tableau rules given in Figure 21 are nondeterministic. Any node marked $\Delta, w \models (\psi_1 \wedge \psi_2)$ has two children, where one is marked $\Delta, w \models \psi_1$ and the other $\Delta, w \models \psi_2$. For a node marked $\Delta, w \models (\psi_1 \vee \psi_2)$ there is only one child node which is either marked $\Delta, w \models \psi_1$ or $\Delta, w \models \psi_2$. Thus, for a given point w and formula φ, there are several nonequivalent completed tableaus; $w \models \varphi$ iff *some* of these tableaus is successful. A tableau is successful, if *each* leaf is successful. To turn the tableau method into a concrete model checking algorithm, we have to perform a depth-first search through all possible tableaus.

$$(\vee_i)\ \frac{\Delta, w \models (\psi_1 \vee \psi_2)}{\Delta, w \models \psi_i}\ {\scriptstyle (i \in \{1,2\})} \qquad (\wedge)\ \frac{\Delta, w \models (\psi_1 \wedge \psi_2)}{\Delta, w \models \psi_1 \qquad \Delta, w \models \psi_2}$$

$$(\langle R\rangle)\ \frac{\Delta, w \models \langle R\rangle\,\psi}{\overline{\Delta, w' \models \psi}} \qquad ([R])\ \frac{\Delta, w \models [R]\,\psi}{\overline{\Delta, w_1 \models \psi \quad \cdots \quad \Delta, w_n \models \psi}}$$

$$(\mu)\ \frac{\Delta, w \models \mu q\psi}{\Delta', w \models \psi} \qquad (\nu)\ \frac{\Delta, w \models \nu q\psi}{\Delta', w \models \psi}$$

$$(\text{PVar})\ \frac{\Delta, w \models q}{\Delta, w \models \psi}$$

Figure 21: Tableau rules for branching time μ**TL**

The additional regulations for the tableau rules in Figure 21 are:

- Rule ($\langle R\rangle$) can only be applied if $w' \in R(w)$.
- In rule ($[R]$), it must hold that $R(w) = \{w_1, ..., w_n\}$.
- In rule (μ) and (ν), $\Delta' \triangleq \Delta \cup \{q = \psi\}$.
- Rule (PVar) can only be applied if $(q = \psi) \in \Delta$, and there is no ancestor node which is labelled $\Delta', w \models \psi$ (with the same w and ψ).

Intuitively, to check whether $\mu q\psi$ holds in point w, we record that q must be interpreted as a fixpoint of $\psi(q)$, and check whether ψ holds in w. Whenever we hit upon the proposition variable q in the further decomposition of $\psi(q)$, we can unfold this occurrence to ψ. However, to guarantee that the unfolding terminates, each proposition variable may be unfolded at most once in every branch of the tableau and every point of the model. Thus, for finite models each tableau is finite.

A tableau is maximal, if there is no leaf for which any rule is applicable. In a maximal tableau, a leaf $\Delta, w \models \psi$ is called *successful*, if

- $\psi = \mathtt{p} \in \mathcal{P}$ and $w \in \mathcal{I}(\mathtt{p})$, or $\psi = \neg\mathtt{p}$ and $w \notin \mathcal{I}(\mathtt{p})$,
- $\psi = q \in \mathcal{Q}$, $q \notin \Delta$, $w \in \mathbf{v}(q)$, or $\psi = \neg q$, $q \notin \Delta$, $w \notin \mathbf{v}(q)$, or
- $\psi = [R]\,\psi'$ and $R(w) = \{\}$ (Rule ($[R]$) produces no children),
- $\psi = q \in \mathcal{Q}$ and q was included in Δ by rule (ν).

In other words, a maximal tableau is not successful if it contains some unsuccessful leaf $\Delta, w \models \psi$ which satisfies

- $\psi = \mathtt{p} \in \mathcal{P}$ and $w \notin \mathcal{I}(\mathtt{p})$, or $\psi = \neg\mathtt{p}$ and $w \in \mathcal{I}(\mathtt{p})$,
- $\psi = q \in \mathcal{Q}$, $q \notin \Delta$, $w \notin \mathbf{v}(q)$, or $\psi = \neg q$, $q \notin \Delta$, $w \in \mathbf{v}(q)$, or
- $\psi = \langle R\rangle\,\psi'$ and $R(w) = \{\}$ (Rule ($\langle R\rangle$) not applicable),
- $\psi = q \in \mathcal{Q}$ and q was included in Δ by rule (μ).

With these definitions, soundness and completeness of the tableau decision method is stated in the following theorem, a proof of which can be found in [Stirling and Walker 1991].

8.1. THEOREM. $w \in \varphi^{\mathcal{F}}$ *iff there exists a successful tableau with root* $\{\}, w \models \varphi$.

More efficient local model checking algorithms for fragments of $\mu\mathbf{TL}$ can be found in [Cleaveland and Steffen 1993, Bhat and Cleaveland 1996].

A somewhat different approach for model checking of μ-calculus was suggested in [Mader 1992]. It is based on *Gauss-elimination*: proving a formula in this approach is similar to solving a system of linear inequalities.

9. Modelling of Reactive Systems

Up to now, we assumed that a system is given as a single Kripke-model. However, real-life systems usually are composed of a number of smaller subcomponents. Even if the target system is a single sequential machine, it is often advantageous to model it as a set of processes running in parallel:

- usually the *functionality* suggests a certain decomposition into modules; sequentialization is not the primary issue in the design;
- certain subcomponents (e.g. hardware components) actually are independent of the rest of the system and, therefore, conceptually parallel,
- the *environment* can be seen as a process running in parallel to the system;
- software-reusability and object-oriented design require modularity.

9.1. Parallel Programming Paradigms

Hence, we have to consider systems of *parallel processes*, that is, processes which are executed during the same time period, and the synchronization between these processes. We distinguish between two main paradigms of parallel systems: *distributed* systems, where the subcomponents are seen as spatially apart from each other, and *concurrent* systems, where the subcomponents use common resources such as processor time or memory cells.

Message Passing vs. Shared Variables
Consequently, there are two main paradigms for synchronization between parallel processes: via *message passing* (for distributed systems), and via *shared variables* (for concurrent systems).

Of course, there is no clear distinction between distributed and concurrent programs. It is not possible to formalize the concept of being spatially apart, since this is dependent on one's own point of view: from the United States, all computers in a local area network in Europe can be regarded as a single system. From the processor's viewpoint, a hard disk controller can be regarded as a remote subsystem.

On the other side, every component of a distributed system shares *some* resource with *some* other component; if it were totally unrelated it would not make sense to regard it as being part of one system.

Consequently, from a certain point of view, passing a message between process A and B can be seen as process A writing into a shared variable which is read by B. On the other side, writing a shared variable can be seen as sending to all other processes which might use this variable the message that its value has changed. In fact, this transition from the message passing paradigm to an implementation via shared variables occurs in every network controller; and the transition from the shared variables paradigm to an implementation via message passing occurs in every distributed cache.

However, different paradigms produce different techniques; many parallel programming languages and many verification systems support only one of these two paradigms.

Synchronous vs. Asynchronous Systems

Another issue is the modelling of a process execution in time. In *discrete* processes a computation consists of a sequence of steps, whereas in *continuous* systems the value of state parameters changes gradually as time passes. *Hybrid systems* combine discrete and continuous components. Usually, the model of time which is used in verification is determined by the type of system under consideration.

For parallel systems of discrete processes, there are various ways to model their execution. *Synchronous* processing is characterized by the fact that in each step, every parallel component advances. For example, a circuit in which each gate switches at the pulse of a global clock can be seen as a synchronous system. In contrast, in an *asynchronous* execution in each step an arbitrary (nonempty) subset of all components proceeds. For example, a set of agents working independently and synchronizing via mailboxes is a typical asynchronous system. With synchronous processing, the transition relation of the system is the conjunction of the transition relations of the components, with asynchronous processing it is the disjunction.

If each process can perform an “idle” step at any time (“*stutter*”), then synchronous and asynchronous processing coincides. Both synchronous and asynchronous executions can be implemented by *interleaving*, where in each step at most one process is active. A typical example is a set of threads in a time-sharing operating system on a mono-processor machine. With interleaving execution, usually some fairness constraints are imposed on the scheduling to ensure that all processes can progress.

Related to the execution mode is the mode of interaction between parallel components. With *synchronous communication*, each component wishing to interact is blocked until all partners it requires are willing to participate in the communication. The information is then broadcast to all communication partners. With *asynchronous communication* each process decides whether it wants to wait at a certain point or not; usually some kind of buffering mechanism is used for messages which are not needed immediately.

Synchronous communication can be seen as a special case of asynchronous communication where the length of each buffer queue is limited to one, and each process decides to wait after writing into or before reading from that queue until the queue is empty or full again, respectively.

Vice versa, a buffer can be seen as a separate process in a synchronous system which is always willing to communicate with other processes. If the size of the buffer is unbounded, the system is not finite state. Even if their size is bounded, the buffers can be the biggest part of the modelling of an asynchronously communicating system.

9.2. Some Concrete Formalisms for Finite State Systems

Recall that a (labelled) *transition system* is a tuple (Σ, S, Δ, S_0), where

- Σ is a nonempty finite *alphabet*,
- S is a nonempty finite set of *states*,
- $\Delta \subseteq S \times \Sigma \times S$ is the *transition relation*, and
- $S_0 \subseteq S$ is the set of *initial states*.

A *parallel transition system* is a tuple $T = (T_1, ..., T_n)$ of transition systems, such that $S_i \cap S_j = \{\}$, for $i < j$. The *global* transition system T *associated with* a parallel transition system $(T_1, ..., T_n)$ is defined by $T = (\Sigma, S, \Delta, S_0)$, where

- $\Sigma = \bigcup \Sigma_i$
- $S = S_1 \times \cdots \times S_n$
- $S_0 = S_{10} \times \cdots \times S_{n0}$, and
- $((s_1, ..., s_n), a, (s'_1, ..., s'_n)) \in \Delta$ iff for all T_i
  — if $a \in \Sigma_i$, then $(s_i, a, s'_i) \in \Delta_i$, and
  — if $a \notin \Sigma_i$, then $s_i = s'_i$

Thus, in a parallel transition system synchronization between components is by the common alphabet. The size of the state space of the global transition system is the product of the sizes of all parallel components.

An *elementary Petri net* is a tuple $N = (P, T, F, s_0)$, where

- P is a finite set of *places*,
- T is a finite set of *transitions* $(P \cap T = \{\})$,
- $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*, and
- $m_0 \subseteq P$ is the *initial marking* of the net.

A *marking* m of the net is any subset of P. By $\bullet t \triangleq \{p \mid (p, t) \in F\}$ and $t\bullet \triangleq \{p \mid (t, p) \in F\}$ we denote the *preset* and the *postset* of transition t, respectively. A transition t is *enabled* at marking m if $\bullet t \subseteq m$ (all its input places are occupied at m) and $t\bullet \cap m \subseteq \bullet t$ (all its output places are empty at m, or they are also input places). Marking m' is the *result of firing* transition t from marking m, if t is enabled at m and $m' = (m \setminus \bullet t) \cup t\bullet$. In contrast to condition-event Petri nets [Reisig 1998], where each place can be occupied by an arbitrary number of tokens, elementary Petri nets inherently are finite-state.

For every elementary Petri net there is an associated transition system: the alphabet is the set of transitions, the state set is the set of markings, the initial state is the initial marking, and $(m, t, m') \in \Delta$ iff m' is the result of firing t from m. The number of states of this transition system is exponential in the number of places of the net. Alternatively, for any elementary Petri net we can obtain a parallel transition system of the same order of magnitude: for each place p in the net there is a transition system with two states p^1 and p^0, denoting the fact that p is occupied or empty, respectively. For each $t \in T$, we let $(p^1, t, p^0) \in \Delta$ iff $p \in \bullet t \setminus t\bullet$ and $(p^0, t, p^1) \in \Delta$ iff $p \in t\bullet \setminus \bullet t$. Furthermore, $(p^1, t, p^1) \in \Delta$ iff $p \in \bullet t \cap t\bullet$. The language of the global transition system associated with this parallel transition system is the set of firing sequences of the net. Vice versa, every parallel transition system can be formulated as an elementary Petri net of the same order of magnitude. The construction is straightforward.

A *shared variables program* is a tuple (V, D, T, s_0), where

- $V = (v_1, ..., v_n)$ is a set of *program variables*,
- $D = D_1 \times \cdots \times D_n)$ is the *state space*, where each $D_i = \{d_{i1}, ..., d_{im_i}\}$ is a finite *domain* for variable v_i,
- $T \subseteq D \times D$ is a *transition relation*, and
- $s_0 = (d_{11}, ..., d_{n1})$ is the *initial state*.

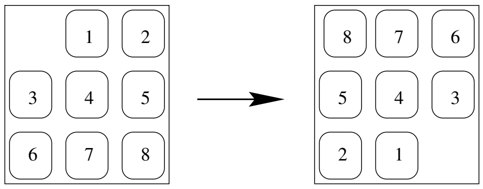
A *state* of a shared variables program is a tuple $(d_1, ..., d_n)$, where each $d_i \in D_i$. Thus the number of states in a shared variables program is the product of the size of all domains. The transition relation T can be defined by a propositional formula φ_T with the set of atomic proposition $\mathcal{P} = \{(x = y) \mid x, y \in (V \cup V' \cup \bigcup D_i)\}$, where $V' = \{v'_1, ..., v'_n\}$. If $s = (d_1, ..., d_n)$ and $s' = (d'_1, ..., d'_n)$, then $(s, s') \in T$ iff $\mathcal{I} \models \varphi_T$, where $\mathcal{I}(v_i) = d_i$ and $\mathcal{I}(v'_i) = d'_i$.

Using relational semantics, a shared variables program can be obtained for almost all other models for concurrency. Therefore, shared variable programs are widely used to model reactive systems.

9.3. Example Applications

A Combinatorial Game

As a first example, we describe the use of model checking in a combinatorial search. Although this example is not very typical for real applications, it can demonstrate the capabilities and limits of present technology. A well-known puzzle from 1870 by the American Sam Loyd consists of a $h \times v$ grid in which there are $(h \cdot v) - 1$ numbered tiles and one blank space. A move consists in moving any tile into the position of the blank. The goal is to achieve a certain predetermined order on the tiles.



This puzzle can be described by a shared variables program as follows. For each tile there is a program variable which notes its horizontal and vertical position. Furthermore, there is a program variable `move` indicating whether the next move will be a shift up, down, left or right of the blank space. If the move would bring it out of the borders, nothing is changed; otherwise, its position is swapped with the respective adjacent tile.

```
MODULE main
DEFINE v := 3; h := 3;
VAR move: u,d,l,r;
    hpos: array 0..(h*v-1) of 1..h;
    vpos: array 0..(h*v-1) of 1..v;
ASSIGN
next(hpos[0]) := case
  (move=l) & !(hpos[0]=1) : hpos[0] - 1;
  (move=r) & !(hpos[0]=h) : hpos[0] + 1;
  1: hpos[0]; esac;
next(vpos[0]) := case
  (move=u) & !(vpos[0]=1) : vpos[0] - 1;
  (move=d) & !(vpos[0]=v) : vpos[0] + 1;
  1: vpos[0]; esac;
for all i:
next(hpos[i]) := case
  (move=l) & !(hpos[0]=1) & vpos[i]=vpos[0] & hpos[i]=hpos[0]+1 |
  (move=r) & !(hpos[0]=h) & vpos[i]=vpos[0] & hpos[i]=hpos[0]-1 : hpos[0];
  1: hpos[i]; esac;
next(vpos[i]) := case
  (move=u) & !(vpos[0]=1) & hpos[i]=hpos[0] & vpos[i]=vpos[0]-1 |
  (move=d) & !(vpos[0]=v) & hpos[i]=hpos[0] & vpos[i]=vpos[0]+1 : vpos[0];
  1: vpos[i]; esac;
init(vpos[i]) := i div h + 1; init(hpos[i]) := i mod h + 1;
DEFINE goal := ⋀i(vpos[i] = v - (i div h) & hpos[i] = h - (i mod h))
SPEC !EF goal
```

Figure 22: SMV Code for Loyds Puzzle

The SMV code corresponding to this description[6] is shown in Figure 22. For $h = 3$ and $v = 3$, the internal representation of the transition relation takes about 3KB. There are $4 \cdot (h \cdot v)! = 1.4 \cdot 10^6$ states, of which 50% are reachable from any

[6] In the actual SMV code, variable array bounds or indices, e.g., vpos[i], are not allowed and have to be replaced by the respective constant values vpos[1],vpos[2],...

initial state. The specification claims that a certain final state is *not* reachable; the model checker contradicts this claim by showing a sequence of moves (rrddlluurrddlluurrddlluurrddlluurrdd) which gives a solution to the puzzle. The solution is found within a couple of minutes on a 32 MB Pentium 133.

For $h = 4$, $v = 3$, there are approximately 10^9 reachable states. Although the symbolic model checker detects rather quickly that some solution must exist, for the construction of a concrete solution sequence the state space has to be partitioned into strongly connected components. This requires several days of CPU time and approximately 1GB RAM on a Sparc Ultra. For model checking applications, virtual memory is not very useful; if the representation of the reachable state space exceeds the available main memory, then constant swapping occurs. To find a solution for $h = 4$, $v = 4$ by exhaustive state space exploration seems to be beyond the limits of present technology. In [Edelkamp and Reffel 1998], a combination of model checking and heuristic search is used to automatically construct solutions to this and other combinatorial games.

A Sequential Circuit

Our second example is from hardware verification. We consider a shift register for interfacing a parallel data bus. The register is from the 74x95 TTL family and is described in [Nowicki and Adam 1990]. It is used to exchange data between the bus and a serial device. It thus acts as parallel-serial converter and vice versa. A functional diagram of the register is given in Figure 23.

The register has a *mode control* input `mc` to choose between parallel or serial access mode. For each mode, there is a corresponding input clock (`pc` and `sc`). Parallel loading is performed if `mc` is high and a `pc` clock pulse arrives. In this case, data is read from the bus into the associated flip-flops. The data appears at the `Q` outputs at the pulse of the `pc` clock.

For serial loading, mode control should be low. Data is input serially with every tick of the `sc` clock. At each pulse the state of all flip-flops is transferred one stage to the right. After n cycles, the data is positioned at the parallel output and can be sent to the bus by an `oc` command. A right shift occurs if the serial input `inp` is held low. By a sequence of n right shifts, data which has been obtained in parallel from the bus can be written serially to the `out` port.

The register is implemented with SR-bistables which have the following characteristic function. If both inputs are low, the bistable keeps its state. The output `Q`

S	R	Q'
0	0	Q
1	0	1
0	1	0
1	1	-

is set if input `S` is high, and reset if input `R` is high. If both `S` and `R` are high, then `Q`

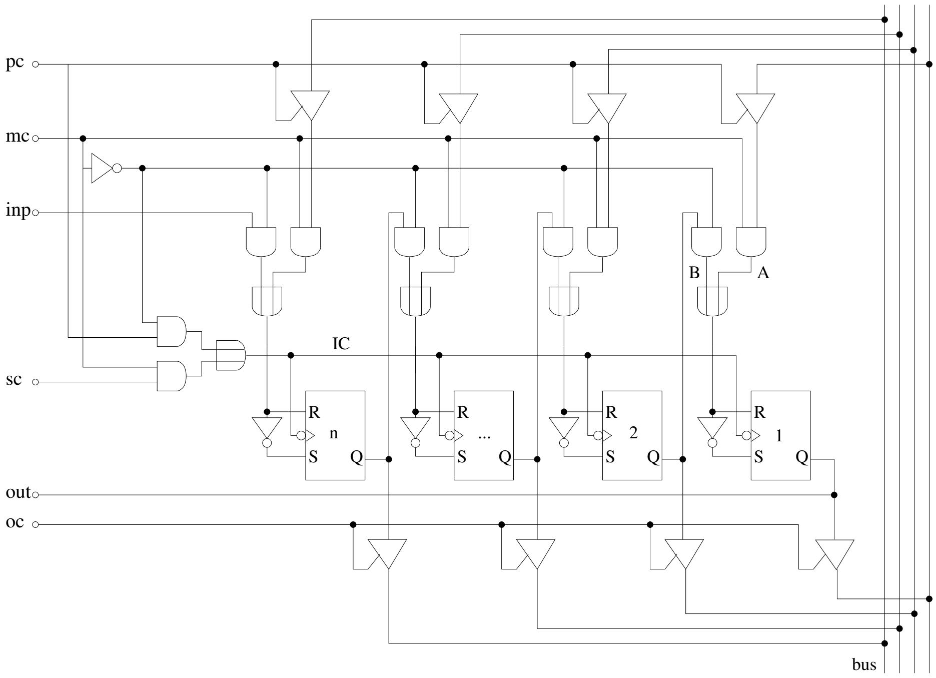


Figure 23: A shift register for data bus interfacing

is undefined. This can be modelled by a nondeterministic internal choice between high and low output. The latch is triggered by a negative edge of the clock pulse. That is, a change of output occurs only at the time instant when the clock line goes from high to low. If the value of the clock line is part of the state space, then the clock value would be low in every new state. For an accurate state-based model (e.g., of an asynchronous circuit), we would have to include timing information of all gates. However, if the clock is only used as trigger, an event based modelling is more adequate: the high-to-low change of the clock line is considered as an event occurrence. In each state, this event may or may not occur. To prevent executions in which the input or output clocks are indefinitely blocked, we require infinitely many input and output clock ticks in every infinite run.

The model is just a representation of the circuit's truth table, where the outputs are a boolean function of inputs and latch states. It can be derived automatically from any standard hardware description language; in fact, several model checkers support such front-end translations. Correctness of parallel and sequential input is expressed by the following formulas, where n is the width of the data bus:

$$\mathbf{A}\,\mathbf{G}^{*}(\mathtt{mc} \wedge \mathtt{pc} \rightarrow \bigvee_{\mathtt{i}=1}^{n} (\mathtt{bus[i]} \leftrightarrow \mathbf{A}((\mathtt{oc} \rightarrow \mathbf{A}\,\mathbf{X}\,\mathtt{bus[i]})\,\mathbf{U}^{+}\,\mathtt{ic})))$$

```
MODULE main
VAR Q, bus: array 1..n of boolean;  -- n SR-latches, n databits
    inp, mc, pc, sc, oc: boolean;   -- input lines
DEFINE out := Q[1]; ic := ((mc & pc)  |  (!mc & sc));
       A[i] := mc & pc & bus[i]; B[i] := !mc & Q[i+1];
       R[i] := !(A[i] | B[i]); S[i] := !R[i];
ASSIGN next(Q[i]) := case ic: case
                                !S[i] & !R[i]: Q[i];        --hold
                                 S[i] & !R[i]: 1;           --set
                                !S[i] &  R[i]: 0;           --reset
                                 S[i] &  R[i]: {0,1}; esac; --undef
                           !ic: Q[i]; esac; -- unchanged if no input
       next(bus[i]) := case oc: Q[i];  !oc: {0, 1}; esac;
FAIRNESS ic FAIRNESS oc
```

Figure 24: Model of shift register

$$\mathbf{A}\,\mathbf{G}^{*}(\neg\mathtt{mc} \wedge \mathtt{sc} \rightarrow \bigvee_{\mathtt{i}=2}^{n} (\mathtt{Q[i]} \leftrightarrow \mathbf{A}(\mathtt{Q[i-1]}\ \mathbf{U}^{+}\ \mathtt{ic})))$$

Intuitively, these formulas assure that data which is input into the register remains there until a new input occurs. If the mode control is set to parallel and there is a tick of the parallel clock, then the data which is currently on bus i will be delivered at each tick of the output clock, until a new input occurs. If the mode control is set to serial, and there is a tick of the serial clock, then the latches will remain stable until the next input.

The SMV model checker can verify these formulas for a bus width of 32 bit in less than a second. Similar formulas can be used to verify that after a sequence of n sequential load operations, the correct data word will be put onto the bus on a subsequent output pulse.

If the connection structure of wires within the circuit is "well-behaved", then automatic verification is successful even on much bigger circuits. A circuit is "well-behaved" if there exists an ordering of all wires such that the value of a wire only depends on the value of wires which are close in the ordering. For a formal definition of this condition see [McMillan 1993]. A large number of circuits with hundreds of storage places have been verified automatically in this way.

A Communication Protocol

The third example is a set of communicating processes within the operating system of a Siemens cellular phone. In this system, there are a number of basic processes communicating with one another by priority messages. Each of the processes implements a finite state machine, which is described by a set of SDL diagrams. Basically, a process waits in a certain state until it receives a message from some other process. It then performs some specified operations, sends a number of messages to other messages, and transitions to another state. Figure 25 shows part of the transition graph of a process and the corresponding SDL diagram. The displayed part is used to implement the following quote from the GSM international standard.

> *"Initially the mobile station looks for a cell which satisfies the suitability constraints by checking cells in descending order of received signal strength. If a suitable cell is found, the mobile station camps on it and performs any registration necessary."*

A property to be verified is that the system never deadlocks:

$$\mathbf{A}\,\mathbf{G}^{*}\,\mathbf{E}\,\mathbf{F}^{*}\,\texttt{init}$$

That is, no sequence of user actions can bring the phone into a state from where it cannot be reset. Since the number of merchandised units is expected to be very high, correctness is an important design issue. In this particular example, a number of potential problems in the design could be identified by model checking before the actual implementation took place [Schlingloff 1997].

In the model to be checked, there are five basic processes, plus the operating system kernel. There are approximately 50 different types of messages which can be sent by the processes, and each process has between 10 and 20 states. The operating system is responsible for the scheduling of processes according to a priority scheme, and for the storage and delivery of messages. Therefore, it has to maintain a buffer, in which for each process all messages are kept. The size of these buffers turns out to be the most important parameter in the verification. Basically, each buffer slot could be filled with every message; thus a combinatorial explosion similar to the one in our first example can occur. However, a buffer overflow almost certainly indicates an error in the implementation; for example, if some high-priority process keeps resending the same message, it will eventually fill up any bounded buffer. In the modelled system, a total number of 15-20 buffer slots was sufficient; a fairness assumption is used to select only those computations in which no buffer overflow occurs. Moreover, the buffer contents usually follows a regular pattern, therefore the above mentioned state explosion is avoided. In practical applications, an exponential growth in the number of reachable states almost certainly indicates an error. For buffers in which all messages have the same priority, the transition relation of a bounded buffer can be defined by the transition table in Figure 26.

In the right half of this table, an empty entry means that the respective program variable is set by the environment. An input value of nil in i indicates that there is no message to be sent; in this case the next value of i is determined by the sender. If this process has put a non-nil value x into i, then this value is appended to the buffer, and i is reset to nil. The last line indicates a buffer overflow: if a message is to be sent with the message buffer already filled, i remains stable. Thus, the formula $\mathbf{A}\,\mathbf{G}^{*}(i \neq nil \rightarrow \mathbf{X}(i = nil))$ can be used to determine whether a buffer overflow can occur. If the output variable o is nil and there is a message to deliver, it is copied into o. When the operating system delivers a message y from o, it resets o to nil.

The content of the buffer b is given as a sequence $\langle x_1, ..., x_\nu \rangle$ of messages, where $\langle\rangle$ denotes the empty buffer. There are various possibilities to model such sequences. In Figure 27 we show a modelling which uses n program variables $b_1, ..., b_n$, such

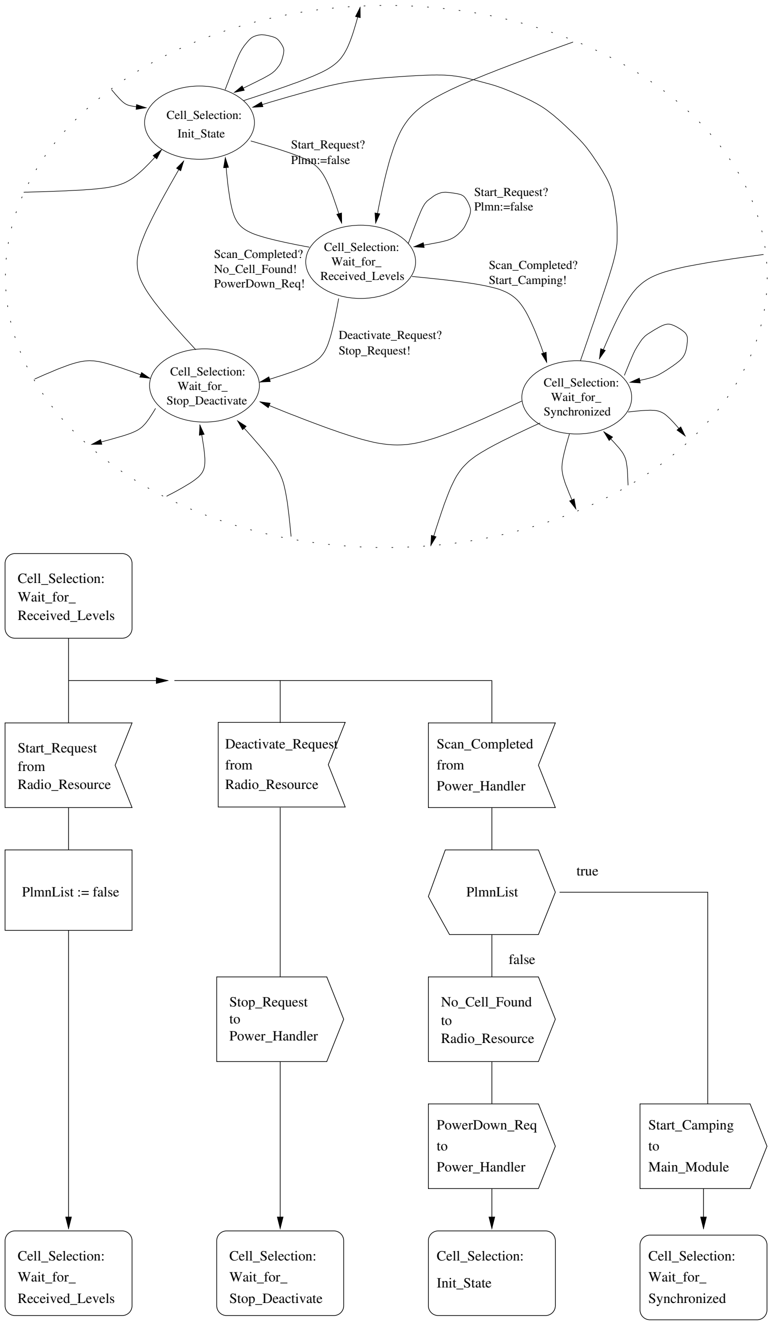


Figure 25: Transition graph and SDL diagram

i	b	o		i'	b'	o'
nil	$\langle\rangle$	nil			$\langle\rangle$	nil
x	$\langle\rangle$	nil		nil	$\langle\rangle$	x
nil	$\langle x_1, ..., x_\nu \rangle$	nil			$\langle x_1, ..., x_{\nu-1} \rangle$	x_ν
x	$\langle x_1, ..., x_\nu \rangle$	nil		nil	$\langle x, x_1, ..., x_{\nu-1} \rangle$	x_ν
nil	$\langle\rangle$	y			$\langle\rangle$	
x	$\langle\rangle$	y		nil	$\langle x \rangle$	
nil	$\langle x_1, ..., x_\nu \rangle$	y			$\langle x_1, ..., x_\nu \rangle$	
x	$\langle x_1, ..., x_\nu \rangle$	y	$(\nu < n)$	nil	$\langle x, x_1, ..., x_\nu \rangle$	
x	$\langle x_1, ..., x_n \rangle$	y		x	$\langle x_1, ..., x_n \rangle$	

Figure 26: Transition relation of a bounded buffer

that b_1 contains the front element of the message queue, and incoming messages are appended into the smallest b_ν which is empty (contains nil as value).

```
next(b[j]) := case
   (i=nil) & !(o=nil) : b[j];
   (i=nil) &  (o=nil) : b[j+1];
  !(i=nil) & !(o=nil) : if !(b[j-1]=nil) & b[j]=nil then i
                        else b[j] fi;
  !(i=nil) &  (o=nil) : if b[j]=nil then nil
                        else if b[j+1]=nil then i
                        else b[j+1] fi fi;            esac;
```

Figure 27: Model of bounded buffer

In this modelling, we rely on the fact that whenever $b_j = nil$, then for all $k \geq j$, also $b_k = nil$. This assumption only holds for the reachable states of a buffer which is initially empty; there are many transitions from illegal, i.e., non reachable states to other illegal states in this model. In an explicit representation of the transition relation, one should try to avoid these redundant entries. Below, we discuss symbolic representations with BDDs. With such a representation, even though the size of the transition relation is much bigger than the transition relation restricted to the reachable states, its representation is much smaller. Since the value of each buffer slot depends only on its immediate neighbors, in fact the size of the representation is linear in the (fixed) number and width of the buffer slots. For modelling unbounded queues, efficient data structures are discussed in [Boigelot and Godefroid 1996, Godefroid and Long 1996].

10. Symbolic Model Checking

Model checking methods derive a great deal of their success from the efficiency of the data structures that are used. A propositional formula can be regarded as a boolean function, mapping an interpretation of the propositions into $\{\mathbf{true}, \mathbf{false}\}$. Since very powerful techniques exist for manipulation of such functions, it makes sense to represent temporal and predicate logic formulas as well as frames in terms of boolean functions. The general idea is to *encode* each domain element by a boolean sequence. Predicates and relations are then represented by their characteristic functions. Temporal operators are interpreted algorithmically according to their fixpoint definitions.

For any shared variables program, we can obtain an equivalent shared variables program which uses only binary domains of the form $D = \{0,1\}^n$. To do so, we use an arbitrary binary encoding of domain D_i and introduce for any program variable v_i over domain D_i new binary program variables $v_{i1}, ..., v_{ik}$, where $k = \lceil \log_2(|D_i|) \rceil$. This encoding is comparable to the implementation of arbitrary data types on digital computers, where each bit can take only two values.

If all program variables $V = \{v_1, ..., v_n\}$ of a shared variables program are over a binary domain, then any propositional formula φ over $\mathcal{P} = \{v_1, ..., v_n\}$ describes a set of states of the program, namely the set of all propositional models (interpretations) which validate the formula. Here we assume the substitution 0 for **false** and 1 for **true**. Vice versa, for any set of states there is a propositional formula describing this set. However, this formula is not uniquely determined; the problem of finding a shortest formula describing a given set of states is co-NP-hard.

The transition relation of a shared variables program with binary program variables $V = \{v_1, ..., v_n\}$ can be represented as an ordinary propositional formula over $\mathcal{P} = \{v_1, ..., v_n, v_1', ..., v_n'\}$. If the transition relation is given as a propositional formula with equalities, we replace 0 by $\bot$, and 1 by $\top$, and $(v = v')$ by $(v \leftrightarrow v')$[7]. For example, the formula

$$v_1 = 0 \to ((v_1' = 1) \wedge (v_2' = v_2) \wedge (v_3' \neq v_3))$$

in this notation becomes

$$\neg v_1 \to (v_1' \wedge (v_2' \leftrightarrow v_2) \wedge \neg(v_3' \leftrightarrow v_3))$$

For a shared variables program with n program variables over binary domains the size of the state space is 2^n. Therefore e.g. the state space of a buffer of length 10 with values between 1 and 1000 is $2^{100} \simeq 10^{30}$. The *reachable* state space is a subset of this state space, which can be of the same order of magnitude. The transition relation for this buffer consists of pairs of states and therefore has a size of approximately 10^{60}.

To perform global model checking on systems of this or bigger size, we need an efficient representation of large sets.

[7] Recall that $\bot$ and $\top$ are propositional formulas, **false** and **true** are truth values and 0 and 1 are domain elements.

10.1. Binary Decision Diagrams

Clearly, a set could be represented by a table of boolean values. Containment of an element in such a set could then be calculated by selecting the appropriate element from the table. Another possible representation of a set is the explicit enumeration of its elements, e.g., as a list or array. However, these representations can be rather wasteful, since they pay no respect to the internal structure of the set. For example, given the domain $D = \{0, 1, ..., 15\}$, the explicit enumeration of the set "all numbers which are even or bigger than 11" is

$$S = \{0, 2, 4, 6, 8, 10, 12, 13, 14, 15\}$$

The bitstring representation is

$$S = (1010101010101111).$$

These representations take $O(|D| \cdot \lceil \log_2(|D|) \rceil)$ memory bits. Bitstrings provide extremely efficient (constant-time) access. In model checking applications, however, the space used by the data is usually more important than the execution time. So, it is desirable to have a concise data structure for representing large sets which still permits efficient access to the elements.

Given a binary encoding $\vec{v} = v_1 v_2 v_3 v_4$ of the domain D, the above explicit enumeration is

$$S = \{0000, 0010, 0100, 0110, 1000, 1010, 1100, 1101, 1110, 1111\}$$

This description corresponds to a propositional formula in disjunctive normal form. A much more succinct representation of the same set can be given by the formula

$$S = \{\vec{v} \mid v_4 = 0 \vee v_1 = 1 \wedge v_2 = 1\}$$

Usually it is hard to find a minimal propositional formula describing a given set of elements. Therefore attention is restricted to formulas in some normal form. A *binary decision diagram* (BDD, [Bryant 1986, Bryant 1992]) is such a canonical form for a propositional formula. BDDs often are substantially more compact than traditional normal forms such as conjunctive or disjunctive normal form, and they can be manipulated and evaluated very efficiently. Hence, they have become widely used for a variety of applications in computer-aided design applications. Many present tools in symbolic simulation and verification of combinational logic and sequential circuits use a BDD library for manipulating large sets. The size of the BDD depends more on the *structure* of the represented set than on its cardinality. For example, the BDD representation of the empty set and the full set are both of constant size one. Because of this dependence on the structure of the represented object, the description of a system with BDDs is sometimes called a *symbolic representation*, and techniques using BDDs to represent objects are called *symbolic techniques*. Subsequently, we describe symbolic model checking. For an alternative introduction to BDDs and BDD based algorithms in automated theorem proving, see [Moore 1994].

The use of BDDs in checking language containment for ω-automata is described in [Touati, Brayton and Kurshan 1991].

In model checking, binary decision diagrams are a preferred datatype for the representation of propositional formulas. They can be understood as an efficient implementation of binary decision trees. Usually, the BDD is much more succinct than the original decision tree. Efficiency is gained by sharing of subtrees and by elimination of unnecessary nodes.

Consider a three-place boolean connective Ite ("if-then-else"), such that

$$\mathsf{Ite}(\varphi, \psi_1, \psi_2) \triangleq ((\varphi \to \psi_1) \wedge (\neg\varphi \to \psi_2)).$$

Equivalently, $\mathsf{Ite}(\varphi, \psi_1, \psi_2) \leftrightarrow ((\varphi \wedge \psi_1) \vee (\neg\varphi \wedge \psi_2))$. Then $(\varphi \to \psi) \leftrightarrow \mathsf{Ite}(\varphi, \psi, \top)$, hence all boolean operators can be expressed with Ite, $\bot$ and $\top$. A formula ψ is said to be in *tree form*, if $\psi = \bot$, or $\psi = \top$, or $\psi = \mathsf{Ite}(v, \psi_1, \psi_2)$, where $v \in \mathcal{P}$ and ψ_1 and ψ_2 are in tree form. In other words, a formula ψ is in tree form, if it uses only Ite, $\bot$, $\top$, and propositions, and, additionally, for every subformula $\mathsf{Ite}(\varphi, \psi_1, \psi_2)$ of ψ, the formula φ is an (atomic) proposition, and ψ_1 and ψ_2 are not propositions. A tree form formula can be drawn as *binary decision tree*, where for each subformula $\mathsf{Ite}(v, \psi_1, \psi_2)$ there is a node labelled v which has ψ_2 and ψ_1 as left and right child nodes, respectively.

Assume a linear ordering $<$ on the set $\mathcal{P}$ of propositions. A tree form formula is said to be in *ordered tree form*, if for every subformula $\mathsf{Ite}(v_1, \varphi_1, \varphi_2)$ of φ, and every subformula $\mathsf{Ite}(v_2, \psi_1, \psi_2)$ of φ_1 or φ_2, it holds that $v_1 < v_2$. An ordered tree form formula is called *reduced*, if it does not contain any redundant subformula $\mathsf{Ite}(v, \psi, \psi)$ (with equal second and third argument). The sequence of leaves of the formula tree in a reduced ordered tree form formula is called the *logical spectrum* of the formula. For any given ordering, the reduced ordered tree form is a normal form. That is, for every propositional formula there is exactly one equivalent formula in reduced ordered tree form. This formula can be obtained by repeated application of the so-called *Shannon expansion*:

$$\varphi \leftrightarrow \mathsf{Ite}(v, \varphi\{v := \top\}, \varphi\{v := \bot\}),$$

and boolean reductions like $\mathsf{Ite}(v, \psi, \psi) \leftrightarrow \psi$ and $(\bot \to \top) \leftrightarrow \top$.

For example, truth table and tree form formula for the above set are given in Figure 28. The reader should also compare the tree form formula to the tree given on the following page.

The reduced ordered tree form formula for the ordering (v_1, v_2, v_3, v_4) of propositions is obtained by repeatedly replacing every redundant subformula $\mathsf{Ite}(v, \psi, \psi)$ in the above tree form formula by ψ:

$$S = \mathsf{Ite}(v_1, \mathsf{Ite}(v_2, \top, \mathsf{Ite}(v_4, \bot, \top)), \mathsf{Ite}(v_4, \bot, \top))$$

In a reduced ordered tree form formula, there might be several identical subformulas. In order to further reduce the length of the formula, we introduce names for subformulas. An *abbreviated* formula is a formula over the extended alphabet $\mathcal{P}_0 \triangleq \mathcal{P} \cup \{\delta_1, ..., \delta_n\}$, together with a (nonrecursive) list of *abbreviations*

v_1	v_2	v_3	v_4	S
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1
1	1	0	0	1
1	0	1	1	0
1	0	1	0	1
1	0	0	1	0
1	0	0	0	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	0
0	1	0	0	1
0	0	1	1	0
0	0	1	0	1
0	0	0	1	0
0	0	0	0	1

$$
\begin{aligned}
S = \mathsf{Ite}(&v_1,\\
&\mathsf{Ite}(v_2,\\
&\quad \mathsf{Ite}(v_3,\\
&\qquad \mathsf{Ite}(v_4, \top, \top),\\
&\qquad \mathsf{Ite}(v_4, \top, \top)),\\
&\quad \mathsf{Ite}(v_3,\\
&\qquad \mathsf{Ite}(v_4, \bot, \top),\\
&\qquad \mathsf{Ite}(v_4, \bot, \top))),\\
&\mathsf{Ite}(v_2,\\
&\quad \mathsf{Ite}(v_3,\\
&\qquad \mathsf{Ite}(v_4, \bot, \top),\\
&\qquad \mathsf{Ite}(v_4, \bot, \top)),\\
&\quad \mathsf{Ite}(v_3,\\
&\qquad \mathsf{Ite}(v_4, \bot, \top),\\
&\qquad \mathsf{Ite}(v_4, \bot, \top))))
\end{aligned}
$$

Figure 28: Truth table and tree form formula

$(\delta_1 \triangleq \psi_1, \ \ldots, \ \delta_n \triangleq \psi_n)$. In each abbreviation, ψ_i is an abbreviated formula $\mathsf{Ite}(v, \varphi, \varphi')$ over the alphabet $\mathcal{P}_i \triangleq \mathcal{P} \cup \{\delta_{i+1}, \ldots, \delta_n\}$. The introduction of names for subformulas is comparable to the introduction of pointers in formula trees: an abbreviated formula can be drawn as a dag (directed acyclic graph), where each node represents a subformula or abbreviation. A formula is *maximally abbreviated*, if

1. no compound subformula $\mathsf{Ite}(v, \varphi_1, \varphi_2)$ appears twice, and
2. no two abbreviations have the same right hand side.

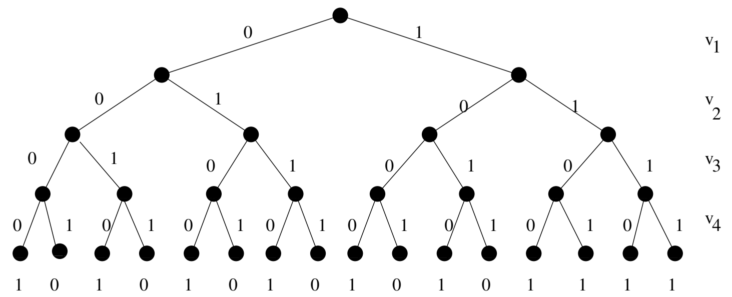
For the above example, a maximally abbreviated formula is

$$S = \mathsf{Ite}(v_1, \mathsf{Ite}(v_2, \top, \delta), \delta), \text{ where } \delta \triangleq \mathsf{Ite}(v_4, \bot, \top)$$

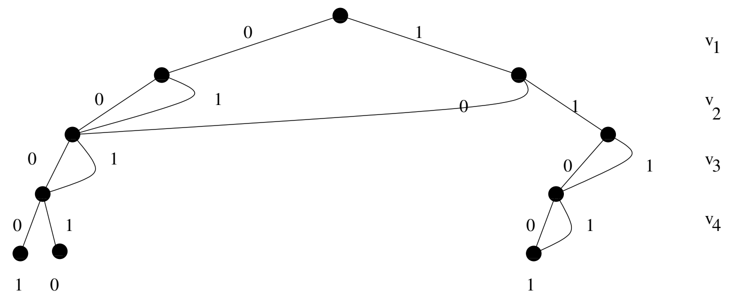
In an implementation an abbreviation can be a pointer or array index to the corresponding subformula. A maximally abbreviated formula is in *BDD form*, if for all subformulas $\mathsf{Ite}(v, \varphi, \psi)$, both φ and ψ are from $\{\bot, \top, \delta_1, \ldots, \delta_n\}$. In the example, this normal form can be obtained by introducing further definitions:

$$S = \mathsf{Ite}(v_1, \delta_1, \delta_2), \text{ where } \delta_1 \triangleq \mathsf{Ite}(v_2, \top, \delta_2) \text{ and } \delta_2 \triangleq \mathsf{Ite}(v_4, \bot, \top)$$

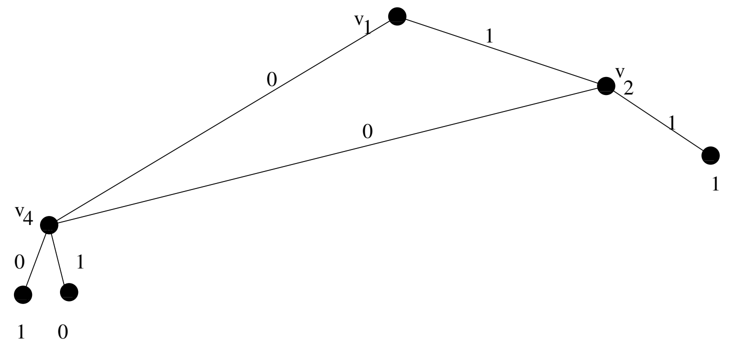
Actually, a BDD form formula is given by a list of abbreviations $(\delta_i \triangleq \mathsf{Ite}(v, \varphi_i, \psi_i))$ and an entry point to this list. It can be drawn as a *binary decision diagram*: for any $\delta \triangleq \mathsf{Ite}(v, \delta_1, \delta_2)$, draw a node labelled v with reference δ, which has the nodes referenced by δ_2 and δ_1 as left and right children, respectively. To illustrate these ideas with pictures, we give the binary decision tree for the above example S:



This tree is just a transcription of the truth table of S's characteristic function. It has many isomorphic subtrees. For any two isomorphic subtrees it is sufficient to maintain only one copy. We can replace the other one by a link to the corresponding subtree.



In the resulting structure, there are nodes for which both alternatives lead to the same subtree. These nodes represent redundant decisions and can be eliminated.



The resulting graph is the (ordered) binary decision diagram for this set with ordering (v_1, v_2, v_3, v_4). Given a variable ordering, there is a canonical BDD for every formula. It can be constructed using the Shannon expansion in a simple recursive descent:

$$\varphi(v_i ... v_n) \leftrightarrow \mathsf{Ite}(v_i, \varphi\{v_i := \top\}(v_{i+1} ... v_n), \varphi\{v_i := \bot\}(v_{i+1} ... v_n))$$

This gives the unique binary decision tree for the chosen ordering. To obtain the BDD for $\varphi(v_i...v_n)$ we recursively calculate the BDD δ_1 for $\varphi\{v_i := \top\}(v_{i+1}...v_n)$ and δ_2 for $\varphi\{v_i := \bot\}(v_{i+1}...v_n)$. Upon backtrack, a new node $\delta \triangleq \mathsf{Ite}(v_i, \delta_1, \delta_2)$ is added to the BDD. However, we do *not* create a new node if both branches in the recursion are equal (return a common result), or if an equivalent node already exists in the BDD. To check this latter condition, we implement the set of BDD nodes $\delta \triangleq \mathsf{Ite}(v, \delta_1, \delta_2)$ as a hash table from (v, δ_1, δ_2) to δ.

Each entry in the hash-table is a quadruple $(\delta, v, \delta_1, \delta_2)$: pointers to BDD nodes are represented as integer numbers. A BDD is identified by its topmost node, and 0 is a pointer to $\bot$ and 1 is a pointer to $\top$. That is, the type "Bdd" is defined as "Int". Likewise, variable names are represented as integer numbers; for clarity we introduce the type "Bddvar" which is also defined as "Int". Thus, for each BDD node $(\delta, i, \delta_1, \delta_2)$ in the hash table, δ (of type "Bdd") is the number of the BDD node, i (of type "Bddvar") is the number of a BDD variable, and δ_1 and δ_2 (of type "Bdd") are links to other BDD nodes. For each (i, δ_1, δ_2) the hash table returns the pointer δ, if this node exists in the BDD.

The resulting algorithm is given in Figure 29. It takes as input a **PL** formula with $\mathcal{P} = \{v_1, ..., v_n\}$ and calculates the table of BDD nodes and a pointer to the topmost node for the variable ordering $(v_1, ..., v_n)$.

function `PL2BDD` (Formula φ) : (Nodeset, Bdd) =
  /∗ Calculates the BDD of φ
    as a set of nodes and a pointer to the topmost node ∗/
  Nodeset *table* := {}; /∗ *Table of BDD nodes* $(\delta, i, \delta_1, \delta_2)$ ∗/
  Bdd *max* := 1; /∗ *Index of maximal table entry* ∗/
  Bdd *result* := `BDD`(φ,1); /∗ *Index of topmost BDD node* ∗/
  return (*table*, *result*);

function `BDD` (Formula φ, Bddvar i) : Bdd =
  /∗ φ is the current subformula, i is the current BDD variable ∗/
  /∗ Return value is a pointer to the maximal BDD node ∗/
  if $i > n$ **then return** eval(φ) /∗ φ *is a boolean constant* ∗/
  else δ_1 := `BDD`($\varphi\{v_i := \bot\}$, $i+1$); δ_2 := `BDD`($\varphi\{v_i := \top\}$, $i+1$);
    if $\delta_1 = \delta_2$ **then return** δ_1
    elsif $\exists\delta : (\delta, i, \delta_1, \delta_2) \in$ *table* **then return** δ
    else *max* := *max* + 1; *table* := *table* $\cup \{(max, i, \delta_1, \delta_2)\}$; **return** *max*;

Figure 29: Transformation of propositional formulas into BDDs

In the BDD representation of sets, several operations can be performed very efficiently. Checking whether a given element w is contained in a set $W \subseteq U$ is done in time $O(\log |U|)$ by traversing the BDD of W according to the bitstring encoding $\vec{w}$

of w. Addition and deletion of elements as well as union and intersection of sets can be done by recursive descent. We now describe this procedure for the implication. Note that the Ite-operator commutes with other boolean connectives:

$$\begin{array}{rcl}(\mathsf{Ite}(p, \varphi_1, \varphi_2) \to \psi) & \leftrightarrow & \mathsf{Ite}(p, (\varphi_1 \to \psi), (\varphi_2 \to \psi)) \\ (\psi \to \mathsf{Ite}(q, \varphi_1, \varphi_2)) & \leftrightarrow & \mathsf{Ite}(q, (\psi \to \varphi_1), (\psi \to \varphi_2))\end{array}$$

Similar equivalences hold for $\wedge$, $\vee$, etc. We prove only the first one of these equivalences. Recall that $\mathsf{Ite}(p, \varphi_1, \varphi_2)$ is defined by $\mathsf{Ite}(p, \varphi_1, \varphi_2) \leftrightarrow ((p \to \psi_1) \wedge (\neg p \to \psi_2))$.

$$\begin{array}{rcl}(\mathsf{Ite}(p, \varphi_1, \varphi_2) \to \psi) & \leftrightarrow & (((p \wedge \varphi_1) \vee (\neg p \wedge \varphi_2)) \to \psi) \\ & \leftrightarrow & (((\neg p \vee \neg \varphi_1) \wedge (p \vee \neg \varphi_2)) \vee \psi) \\ & \leftrightarrow & ((\neg p \vee \neg \varphi_1 \vee \psi) \wedge (p \vee \neg \varphi_2 \vee \psi)) \\ & \leftrightarrow & ((p \wedge (\varphi_1 \to \psi)) \vee (\neg p \wedge (\varphi_2 \to \psi))) \\ & \leftrightarrow & \mathsf{Ite}(p, (\varphi_1 \to \psi), (\varphi_2 \to \psi))\end{array}$$

□

Given BDDs for φ and ψ, the BDD for $(\varphi \to \psi)$ can be constructed as follows. Since $BDD(\varphi)$ and $BDD(\psi)$ can be either 0, 1, or $\mathsf{Ite}(v, \delta_1, \delta_2)$, there are nine cases which have to be considered. If $BDD(\varphi)$ is 0 or $BDD(\psi)$ is 1, the resulting BDD is

```
function BDD_imp (Bdd φ, ψ) : Bdd =
    /* Calculates the BDD of (φ → ψ) from the BDDs of φ and ψ */
    if φ = 0 or ψ = 1 then return 1
    elsif φ = 1 then return ψ
    elsif ψ = 0 and (φ, i, φ1, φ2) ∈ table_φ
        then return new_node(i, BDD_imp(φ1, 0), BDD_imp(φ2, 0))
    else (φ, i, φ1, φ2) ∈ table_φ and (ψ, j, ψ1, ψ2) ∈ table_ψ
        if i = j then return new_node(i, BDD_imp(φ1, ψ1), BDD_imp(φ2, ψ2))
         elsif i < j then return new_node(i, BDD_imp(φ1, ψ), BDD_imp(φ2, ψ))
        elsif i > j then return new_node(j, BDD_imp(φ, ψ1), BDD_imp(φ, ψ2));

function new_node (Bddvar i, Bdd δ1, δ2) : Bdd =
    /* Returns a pointer to a new or existing BDD node */
    /* i is the number of a BDD variable, δ1, δ2 pointers to BDD nodes */
    if δ1 = δ2 then return δ1
    elsif ∃δ : (δ, i, δ1, δ2) ∈ table then return δ
    else max := max + 1; table := table ∪ {(max, i, δ1, δ2)}; return max;
```

Figure 30: Combination of BDDs

1. If $BDD(\varphi)$ is 1, the resulting BDD is $BDD(\psi)$. If $BDD(\varphi)$ is an internal node $\mathsf{Ite}(v, \delta_1, \delta_2)$, and $BDD(\psi)$ is the leaf 0, we use the equivalence:

$$(\mathsf{Ite}(v, \delta_1, \delta_2) \to \bot) \qquad \leftrightarrow \qquad \mathsf{Ite}(v, (\delta_1 \to \bot), (\delta_2 \to \bot))$$

Since $\neg\varphi \triangleq (\varphi \to \bot)$, this means that the BDD for $\neg\varphi$ is constructed from the BDD for φ by exchanging all leafs 0 and 1. The only remaining case is that both $BDD(\varphi) = \mathsf{Ite}(v, \varphi_1, \varphi_2)$ and $BDD(\psi) = \mathsf{Ite}(v', \psi_1, \psi_2)$ are internal nodes. There are three subcases:

1. $v = v'$: $(\mathsf{Ite}(v, \varphi_1, \varphi_2) \to \mathsf{Ite}(v, \psi_1, \psi_2)) \leftrightarrow \mathsf{Ite}(v, (\varphi_1 \to \psi_1), (\varphi_2 \to \psi_2))$
2. $v < v'$ in the order of variables:
   $$(\mathsf{Ite}(v, \varphi_1, \varphi_2) \to \mathsf{Ite}(v', \psi_1, \psi_2)) \\ \leftrightarrow \mathsf{Ite}(v, \varphi_1 \to \mathsf{Ite}(v', \psi_1, \psi_2), \varphi_2 \to \mathsf{Ite}(v', \psi_1, \psi_2))$$
3. $v > v'$ in the order of variables:
   $$(\mathsf{Ite}(v, \varphi_1, \varphi_2) \to \mathsf{Ite}(v', \psi_1, \psi_2)) \\ \leftrightarrow \mathsf{Ite}(v', \mathsf{Ite}(v, \varphi_1, \varphi_2) \to \psi_1, \mathsf{Ite}(v, \varphi_1, \varphi_2) \to \psi_2)$$

In all of these subcases, the BDD for $(\varphi \to \psi)$ is constructed by a recursive call according to the indicated equivalence. Again, upon backtrack a new node is created only if both links are different and no equivalent node exists so far. The algorithm is given in Fig. 30. Some BDD implementations use negated edges to avoid the recursive descent for $\neg\varphi$. Other implementations hash subformulas, such that certain recursive descents can be avoided all together. For more information, see [Brace, Rudell and Bryant 1990].

The complexity of the function `BDD_imp` is linear in the size of the argument BDDs. In principle, all 16 two-argument boolean operations on BDDs can be implemented with linear complexity via this procedure. For example, the BDD for the intersection of two sets φ and ψ can be calculated from the BDDs of φ and ψ using the definition $(\varphi \wedge \psi) \leftrightarrow \neg(\varphi \to \neg\psi)$. In practice, however, most BDD libraries achieve a better performance by providing for each connective a special recursive procedure which takes symmetries and idempotences in the arguments into respect. [Bryant 1986] gives a uniform scheme to handle all 16 boolean connectives. In Fig. 31 this generic `BDD_apply` function is given; the idea of using a co-factoring function is from the BDD library by D. Long.

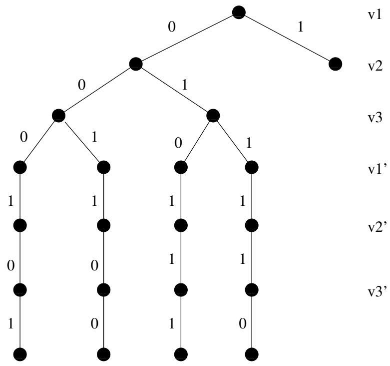
For a given boolean function, the size of the BDD depends critically on the ordering of the variables. For the example formula above (cf. page 10)

$$v_1 = 0 \to ((v_1' = 1) \wedge (v_2' = v_2) \wedge (v_3' \neq v_3))$$

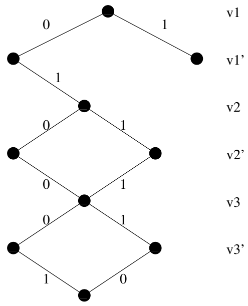
and the variable ordering $(v_1, v_2, v_3, v_1', v_2', v_3')$, the above algorithm yields the following BDD. (We omit all branches leading to negative leaves.)

```
function BDD_apply (Fun ∘, Bdd φ, ψ) : Bdd =
    /* Calculates the BDD of (φ ∘ ψ) from BDDs of φ and ψ */
    if φ ∈ {0,1} and ψ ∈ {0,1} then return φ ∘ ψ
    else m := min_var(φ, ψ);
        (f0, f1) := co_factor(φ, m);   (g0, g1) := co_factor(ψ, m);
        δ1 := BDD_apply(∘, f0, g0);   δ2 := BDD_apply(∘, f1, g1);
        return new_node(m, δ1, δ2);

function min_var (Bdd φ, ψ) : Bddvar =
    /* Returns the minimal BDD variable in φ and ψ */
    if φ ∈ {0,1} and (ψ, j, ψ1, ψ2) ∈ table then return j
    elsif (φ, i, φ1, φ2) ∈ table and ψ ∈ {0,1} then return i
    elsif (φ, i, φ1, φ2) ∈ table and (ψ, j, ψ1, ψ2) ∈ table then return min(i, j);

function co_factor (Bdd δ, Bddvar m) : (Bdd, Bdd) =
    /* Returns two BDD pointers to combine */
    if δ ∈ {0,1} then return (δ, δ)
    else /* (δ, i, δ1, δ2) ∈ table */
        if i > m then return (δ, δ) else return (δ1, δ2);
```

Figure 31: Applying arbitrary functions to BDDs



For the variable ordering $(v_1, v_1', v_2, v_2', v_3, v_3')$, however, we obtain the following much smaller BDD:



This is a common phenomenon when working with BDDs. In general, a good heuristics is to keep "dependent" variables as close together in the ordering as possible [Fuji, Ootomo and Hori 1993, Enders, Filkorn and Taubner 1993]. For a more formal treatment in the context of sequential circuits, see [Bermann 1991, McMillan 1993]. Unfortunately, the problem of finding an *optimal* variable ordering is NP-hard[Bryant 1991]. Basically, for every possible ordering one has to construct the BDD and compare their sizes, which is not feasible. Automatic reordering strategies usually proceed by steepest ascend heuristics [Felt, York, Brayton and Vincentelli 1993, Rudell 1993, Bern, Meinel and Slobodovà 1995].

10.2. Symbolic Model Checking for **CTL**

In [Burch, Clarke, McMillan, Dill and Hwang 1992], the term *symbolic* model checking was introduced for algorithms which use a BDD representation of the Kripke model (cf. Page 1735).

Assume that the transition relation is given as a BDD over the variables $(v_1, ..., v_n, v'_1, ..., v'_n)$, and for each $\mathtt{p} \in \mathcal{P}$ a BDD over $(v_1, ..., v_n)$ is given which represents the set $\mathcal{I}(\mathtt{p})$. We will show how the naïve **CTL** model checking algorithm in Fig. 17 on P. 1714 can be implemented directly with this representation.

Assume that φ is a propositional formula given as a BDD. Substitution $\varphi\{v := b\}$ of a proposition v in φ by a constant value $b \in \{\bot, \top\}$ can be done by assigning a pointer to the appropriate leaf (0 or 1) to each v node. Thus, the function that restricts some argument of a boolean function can be computed in time which is linear in the representation of the function. By using the substitution algorithm, boolean quantification $\exists v\ \varphi$ can be reduced to restriction by

$$(\exists v\ \varphi) \leftrightarrow (\varphi\{v := \bot\} \vee \varphi\{v := \top\})$$

Of course, it would be inefficient to implement simultaneous quantification $\exists \vec{w}\ \varphi$ on a set $\vec{w} \triangleq (v_1 ... v_n)$ of variables by a sequence of such substitutions and disjunctions. Fig. 32 shows how to calculate $\exists \vec{w}\ \varphi$ in a more direct way.

We now describe how to obtain a BDD representation of $\varphi^{\mathcal{F}}$ for any **CTL** formula φ from the given BDD representation of $\mathcal{F}$. The BDDs for $\bot$ and $\mathtt{p} \in \mathcal{P}$ are trivial. The calculation of boolean composites of BDDs was described in the previous subsection. The evaluation of $\mathbf{E}\,\mathbf{U}^{+}$ and $\mathbf{A}\,\mathbf{U}^{+}$ involves computing a fixed

```
function BDD_exists (Set_of_Bddvar w, Bdd φ) : Bdd =
    /* w = {w1...wn} is a set of BDD variables, φ the BDD of a formula */
    /* Result is a BDD for ∃w1...∃wn φ */
    if φ ∈ {0,1} then return φ
    else /* (φ, i, φ1, φ2) ∈ table */
        δ1 := BDD_exists(w, φ1);   δ2 := BDD_exists(w, φ2);
        if i ∈ w then return BDD_apply(or, δ1, δ2)
        else return new_node(i, δ1, δ2);
```

Figure 32: Boolean quantification on BDDs

point. This is done according to the iteration given in Figure 17. In the evaluation of $\mathbf{E}(\psi_2\,\mathbf{U}^+\,\psi_1)$, we have to build the set $\{w \mid \exists w'(w \prec w' \wedge w' \in (\psi_1^{\mathcal{F}} \cup \psi_2^{\mathcal{F}} \cap E))\}$, where E is an intermediate result of the iteration. This formula is an instance of the scheme $\{w \mid \exists w'(\varphi(w') \wedge \psi(w,w'))\}$. Assume we are given a BDD for φ defined over the variables $\vec{w} \triangleq (v_1,...,v_n)$, and a BDD for ψ in the variables $(v_1,...,v_n,v_1',...,v_n')$. The BDD for $\exists w'(\varphi(w') \wedge \psi(w,w'))$, which uses variables $(v_1,...,v_n)$, can be obtained as follows. We first rename all variables v_i in the BDD for φ by v_i'. Then we intersect this BDD with the BDD for ψ to obtain a BDD over $(v_1,...,v_n,v_1',...,v_n')$. Finally, all primed variables are "thrown away" by existential quantification on $w' \triangleq (v_1',...,v_n')$. The case of $\mathbf{A}(\psi_2\,\mathbf{U}^+\,\psi_1)$, where we have to calculate $\{w \mid \forall w'(w \prec w' \rightarrow w' \in (\psi_1^{\mathcal{F}} \cup \psi_2^{\mathcal{F}} \cap E))\}$, is similar.

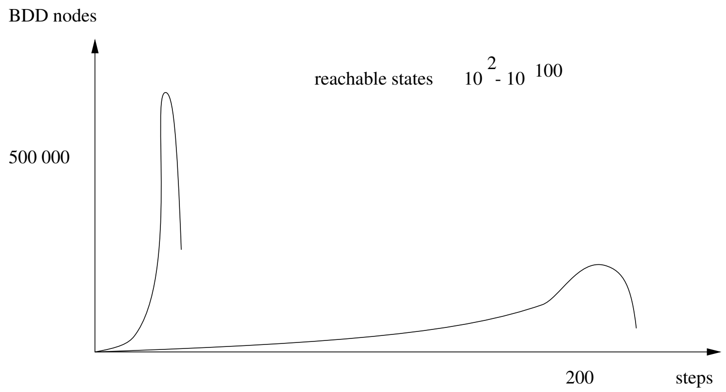
In fact, all of the above BDD operations for one iteration step can be performed during a simple BDD traversal, if v_i and v_i' are always kept together in the variable order. This so-called *relational product algorithm* is similar to the `BDD_apply` and `BDD_exists` algorithms in Figs. 31 and 32. Assume that we are given BDD representations of φ and ψ, where the variable ordering in the BDD for φ is $w_1...w_n$ and in ψ it is $v_1...v_{2n}$, where $w_i = v_{2i-1}$ and $w_i' = v_{2i}$. Function `relprod_BDD` in Fig. 33 calculates the representation of $\exists\vec{w'}(\varphi\{\vec{w} := \vec{w'}\} \wedge \psi)$. The result contains BDD variables $v_1v_3...v_{2n-1}$; renaming to $w_1...w_n$ can be done whenever a new node is created ($v_i = w_{i+1/2}$).

In theory, the complexity of the **CTL** model checking algorithm based on BDDs is not better than with an explicit representation. In practice, however, the BDD representation of large sets of points in realistic systems tends to be quite manageable. Moreover, the number of iteration steps required to reach a fixed point is often small ($\leq 10^3$). For hardware systems, that is, in the verification of sequential circuits, most states are reachable in very few steps, but the BDDs tend to grow exponentially in the first few steps. For software systems, especially if there is not much parallelism contained, the BDD often grows only linear with the number of steps, until the whole state space is traversed. The following picture shows the relation between the BDD size and number of steps in typical examples.

function `BDD_relprod` (Bdd φ, ψ) : Bdd =
  /* Calculates a BDD for $\exists w'(\varphi(w') \wedge \psi(w, w'))$ */
  /* φ has variables $1..n$ and ψ has variables $1...2n$ */
  /* Result contains BDD variables $1, 3, 5..., 2n-1$ */
  if $\varphi = 0$ **or** $\psi = 0$ **then return** 0
  elsif $\psi = 1$ **then return** 1
  else $m :=$ `min_var_2`(φ, ψ); /* *Substitution* $\{w := w'\}$ *in* φ */
    $(f_0, f_1) :=$ `co_factor`$(\varphi, m \text{ div } 2)$; $(g_0, g_1) :=$ `co_factor`(ψ, m);
    $\delta_1 :=$ `BDD_relprod`$(f_0,\, g_0)$; $\delta_2 :=$ `BDD_relprod`$(f_1,\, g_1)$;
    if even(m) **then return** `BDD_apply`(or, δ_1, δ_2)
    else return `new_node`(m, δ_1, δ_2);

function `min_var_2` (Bdd φ, ψ) : Bddvar /* *Ass.:* $(\psi, j, \psi_1, \psi_2) \in table_\psi$ */ =
  /* Returns an appropriate variable number for `BDD_relprod` */
  if $\varphi \in \{0, 1\}$ **then return** j
  else /* $(\varphi, i, \varphi_1, \varphi_2) \in table_\varphi$ */ **return** $\min(2 \cdot i, j)$;

Figure 33: Relational product on BDDs $(\exists w'(\varphi(w') \wedge \psi(w, w')))$



10.3. Relational μ-Calculus

The global algorithm for model checking the propositional μ-calculus can be implemented with BDDs similar as the **CTL** algorithm above. The relational product algorithm can be used to calculate each single step in the fixpoint iteration of modal formulas. We now show how this technique can be extended to a richer logical language which is closer to other programming paradigms. We use a relational μ-calculus similar to the one presented in [Park 1974]. In computer science, [Chandra and Harel 1980] were the first to use similar fixed point operators for the specification of queries in relational databases. In contrast to these papers, we do not use function symbols; they could be added easily to this framework as special relations. Informally, the relational μ-calculus can be seen as first order predicate logic with an additional recursion operator. More information on the logical properties of this calculus can be found in [Vardi 1982, Immerman 1986, Gurevich and Shelah 1986, Dawar, Lindell and Weinstein 1996].

A (typed) *structure* S consists of a collection of disjoint sets called *domains*, and a collection of relations over these domains. (In some textbooks, structures are called *algebras*.) Elements of the domains are called *objects*. Models for propositional temporal logics can be regarded to be special structures with a single domain U, unary predicates $P \subseteq U$ and binary relations $R \subseteq U \times U$ on this domain.

A *signature* $\Sigma = (\mathcal{D}, \mathcal{R})$ consists of a finite set $\mathcal{D}$ of *domain names*, and a finite set $\mathcal{R}$ of *relation symbols*. Associated with each relation symbol is its *type* τ, which is a sequence of domain names. Unary relation symbols are called *predicate symbols*.

An *interpretation* $\mathcal{I}$ for a signature Σ on a structure S is a mapping $\mathcal{I} : \Sigma \to S$ assigning a nonempty domain $\mathcal{I}(D)$ for each domain name D and a relation of appropriate arity for each relation symbol. That is, if $\tau(R) = (D_1, ..., D_n)$, then $\mathcal{I}(R) \subseteq (\mathcal{I}(D_1) \times \cdots \times \mathcal{I}(D_n))$. If the interpretation of a predicate symbol P is a singleton set, we say that P is a *constant*.

Given a signature Σ, let $\mathcal{V}$ be a set of variables, each of which is either an *individual variable* or a *relation variable*. Again, we assume that each variable has an appropriate type. In the relational μ-calculus, there are two more syntactic categories: *well-formed formulas* and *relation terms of type* τ. Assuming that the symbols (,), $\perp$, $\to$, $=$, $\exists$, μ and λ are not in the signature, a *well formed formula* φ is built according to the following syntax:

- $\perp$, $(\varphi \to \psi)$, where φ and ψ are well formed formulas,
- $(x_1 = x_2)$, where x_1 and x_2 are individual variables of the same type,
- $\exists x\ \varphi$, where φ is a well formed formula, and x is an individual variable, or
- $\rho\ x_1 ... x_n$, where ρ is a relation term of type $(D_1, ..., D_n)$ (see below), and x_i is an individual variable of type D_i for all $i \leq n$.

In first order logic, a relation term is just a relation symbol from the signature. In second order logic, a relation term can either be a relation symbol or a relation variable $q \in \mathcal{Q}$. In the relational μ-calculus, more complex relations can be specified via λ-*abstraction* and μ-*recursion*. In this calculus, a *relation term* ρ of type $(D_1, ..., D_n)$ is

- a relation symbol R or relation variable X of type $(D_1, ..., D_n)$,
- $\lambda x_1 ... x_n\ \varphi$, where φ is a well formed formula and each x_i is an individual variable of type D_i, or
- $\mu X\ \rho$, where X is a relation variable of type $(D_1, ..., D_n)$, and ρ a relation term of the same type which is positive in X.

As in the propositional case, in this definition ρ is defined to be *positive* in X, if every occurrence of X is under an even number of negation signs. Positiveness ensures that the functional defined by ρ is monotonic in the lattice of values for X and thus the least fixpoint of the functional exists.

A *variable valuation* $\mathbf{v}$ is a mapping assigning an object $\mathbf{v}(x) \in D$ to every individual variable x of type D, and a relation $\mathbf{v}(X) \subseteq D_1 \times \cdots \times D_n$ to every relation variable X of type $(D_1, ..., D_n)$. A *relational model* $\mathcal{M} \triangleq (S, \mathcal{I}, \mathbf{v})$ for the signature Σ consists of a structure S, an interpretation $\mathcal{I}$, and a variable valuation $\mathbf{v}$. Similar to first order and temporal logics, we say that the model $\mathcal{M} \triangleq (S, \mathcal{I}, \mathbf{v})$ is *based* on the *frame* $\mathcal{F} \triangleq (S, \mathcal{I})$. Any relational model $\mathcal{M} \triangleq (S, \mathcal{I}, \mathbf{v})$ determines an object $x^{\mathcal{M}}$ for every individual variable x, a relation $\rho^{\mathcal{M}}$ of appropriate type for each relation term ρ, and a unique truth value $\varphi^{\mathcal{M}} \in \{\mathbf{true}, \mathbf{false}\}$ for any formula φ. This *denotation* of variables and formulas is defined in the usual way:

- $x^{\mathcal{M}} \triangleq \mathbf{v}(x)$, if $x \in \mathcal{V}$ is an individual variable,
- $\bot^{\mathcal{M}} \triangleq \mathbf{false}$,
- $(\varphi \to \psi)^{\mathcal{M}} = \mathbf{true}$ iff $\varphi^{\mathcal{M}} = \mathbf{true}$ implies $\psi^{\mathcal{M}} = \mathbf{true}$,
- $(x_1 = x_2)^{\mathcal{M}} = \mathbf{true}$ iff $x_1^{\mathcal{M}} = x_2^{\mathcal{M}}$; i.e., iff x_1 and x_2 denote the same object in S,
- $(\exists x\ \varphi)^{\mathcal{M}} = \mathbf{true}$ iff $\varphi^{(S,\mathcal{I},\mathbf{v}')} = \mathbf{true}$ for some valuation $\mathbf{v}'$ which differs from $\mathbf{v}$ at most in x,
- $(\rho\ x_1...x_n)^{\mathcal{M}} = \mathbf{true}$ iff $(x_1^{\mathcal{M}}, ..., x_n^{\mathcal{M}}) \in \rho^{\mathcal{M}}$,
- $R^{\mathcal{M}} \triangleq \mathcal{I}(R)$, if R is a relation symbol,
- $X^{\mathcal{M}} \triangleq \mathbf{v}(X)$, if X is a relation variable,
- $(\lambda x_1...x_n\ \varphi)^{\mathcal{M}} \triangleq \{(d_1, ..., d_n) \mid \varphi^{\mathcal{F}}(d_1, ..., d_n) = \mathbf{true}\}$, where $\varphi^{\mathcal{F}}(d_1, ..., d_n) \triangleq \varphi^{(S,\mathcal{I},\mathbf{v}')}$ and $\mathbf{v}'$ differs from $\mathbf{v}$ only in the assignment of d_i to x_i for $1 \le i \le n$; i.e., $(\lambda x_1...x_n(\varphi))^{\mathcal{M}}$ is the relation consisting of all tuples of objects for which φ is $\mathbf{true}$, and
- $(\mu X\ \rho)^{\mathcal{M}} \triangleq \bigcap\{Q \mid \rho^{\mathcal{F}}(Q) \subseteq Q\}$, where $\rho^{\mathcal{F}}(Q) \triangleq \rho^{(S,\mathcal{I},\mathbf{v}')}$, and $\mathbf{v}'$ differs from $\mathbf{v}$ only in $\mathbf{v}'(X) = Q$; i.e., $\mu X(\rho)^{\mathcal{M}}$ is the least fixpoint of the functional $\rho^{\mathcal{F}}$.

The relational operators λ and μ are similar to the operators used in λ-calculus and in denotational semantics. In fact, we could define well formed formulas to be object terms of the special type $\{\mathbf{false}, \mathbf{true}\}$. Relation terms could then be defined as function terms with boolean result, and the λ abstraction builds such a function term from a boolean object term.

The relational μ-calculus extends first order logic in a similar way as the propositional μ calculus extends modal logic. In fact, the standard translation from modal into first order logic can be trivially extended into a standard translation from propositional into relational μ calculus. In addition, the relational μ calculus offers some restricted form of non-monadic second order quantification. It contains classical first-order logic as a sublanguage. Note, however, that in the relational μ-calculus there is no λ-abstraction on relation variables. This would result in a second-order calculus. In contrast to second order logic, there is no μ-calculus formula expressing that domain D is finite [Park 1974]. On the other hand, the minimization operator can be expressed in second order logic similar as in the propositional case (cf. Page 1660):

$$\mu X(\rho)\vec{x} \leftrightarrow \forall X(\forall \vec{x}(\rho\vec{x} \to X\vec{x}) \to X\vec{x})$$

Since the induction axiom for arithmetic can be formulated as a least fixpoint formula, the natural numbers have a categorical theory in the relational μ-calculus (for details, see also [Park 1974]). Therefore, the set of valid formulas is not recursively enumerable, and its expressiveness lies properly in between first and second order logic.

The μ-recursion operator can be used to give recursive definitions of boolean functions, similar to the use of recursion in functional and logic programming. As an example, the addition-relation on natural numbers can be defined from the constant Z (zero) and the successor relation S by $\mu X(\lambda xyz(Zx \wedge y = z \vee \exists uv(Sux \wedge Svz \wedge Xuyv)))$. All recursive functions of arithmetic can be defined in this way; therefore, on infinite domains, the relational μ-calculus has the expressive power of Turing machines. On finite domains, the model checking problem is polynomial in the size of the structure. Therefore, only those functions are definable which can be computed with time complexity polynomial in the size of the structure ([Chandra and Harel 1980]). For a restricted converse of this statement, see [Vardi 1982, Immerman 1986].

Given a finite relational frame $\mathcal{F} \triangleq (S, \mathcal{I})$ and a relational term ρ or formula φ, model checking can be used to determine the denotation $\rho^{\mathcal{F}}$ or $\varphi^{\mathcal{F}}$, respectively. In [Burch, Clarke, McMillan, Dill and Hwang 1992], a symbolic model checking algorithm for the relational μ-calculus is given (see Figure 34). Assume for simplicity that each domain is binary; for non–binary domains the algorithm can be extended by an appropriate encoding. In the frame, the interpretation $\mathcal{I}$ of a relation of type $(D_1, ..., D_n)$ is represented by a BDD with variables $v_1, v_2, ..., v_n$.

A term or formula with free individual variables $x_1, ..., x_m$ is represented as a BDD with additional BDD variables $x_1, ..., x_m$. A relation variable is represented by its name; each BDD node can contain (the name of) a relation variable as one of its successors. In other words, each BDD node is a tuple $(\delta, i, \delta_1, \delta_2)$, where δ is the name of this node, i is a variable from the set $\{v_1, ..., v_n, x_1, ..., x_m\}$, and each δ_j is one of the BDD constants 0 or 1, a name of another BDD node, or the name of a relation variable. Substitution of a relation variable with a relation in a BDD can be done by a simple BDD traversal.

The model checking algorithm is divided into two functions, `BDD_form` and `BDD_term`, which recurse over the structure of the formula and term. `BDD_form` inputs a formula φ and (the BDD representation of) the interpretation $\mathcal{I}$ in frame $\mathcal{F}$, and returns a BDD which is satisfied by a given valuation $\mathbf{v}$ iff $(S, \mathcal{I}, \mathbf{v}) \models \varphi$. The first five cases in the function derive directly from the respective semantic definitions and should require no explanation. The last case, application of a relation term ρ, uses the function `BDD_term`$(\rho, \mathcal{I})$ to find a representation of the relational term ρ (under the interpretation $\mathcal{I}$), then substitutes the argument variables x_1, ..., x_n for the place-holder variables v_1, ..., v_n, producing a BDD which is satisfied iff ρ holds for $x_1, ..., x_n$.

The function `BDD_term` takes as arguments a relational term ρ and the BDD representation of the interpretation $\mathcal{I}$. It returns a BDD which represents the relation term in the manner described above. The first and second case in the definition of `BDD_term`, a relation symbol or relation variable, simply return the BDD representation of the relation in the interpretation or the name of the relation variable, respectively. The third case, λ-abstraction, produces a BDD with variables v_1, ...,

v_n substituted for the variables v_1, ..., v_n. This is the representation for an n-ary relation which holds iff its arguments satisfy the formula φ when assigned to x_1, ..., x_n. The most interesting case is the last: the fixed point operator μ. To find the fixed point of a relational term with respect to a free relation variable X we use the standard technique for finding the least fixed point of a monotonic functional in a finite domain. First we evaluate `BDD_term`$(\rho, \mathcal{I})$ to get a BDD r for ρ. Then we compute the fixed point by a series of approximations X^0, X^1, ..., beginning with the empty relation (which is represented by the BDD constant 0). To compute the BDD X^{i+1} from X^i we substitute all occurrences of the variable X in the BDD r with X^i. Since the domain is finite and ρ is positive in X, the series must converge to the least fixed point (cf. Lemma 5.4 and Section 8.3). Convergence is detected when $X^{i+1} = X^i$. In this case, X^i is the BDD for $\mu X\, \rho$. Note that testing for convergence is easy, since with a hash-table implementation of BDD nodes equality can be determined in constant time (cf. the algorithm in Fig. 29).

The μcke model checker [Biere 1997] is one of the first tools for model checking the relational μ-calculus. For each non-binary domain, an appropriate binary en-

function `BDD_form` (Formula φ, Interpretation $\mathcal{I}$) : Bdd =
  /* Calculates the BDD of formula φ in the interpretation $\mathcal{I}$ */
  case φ **of**
    $x \in \mathcal{V}$: **return** $\mathsf{Ite}(x, 1, 0)$;
    $(x_1 = x_2)$: **return** $\mathsf{Ite}(x_1, \mathsf{Ite}(x_2, 1, 0), \mathsf{Ite}(x_2, 0, 1))$;
    $\perp$: **return** 0;
    $(\varphi_1 \to \varphi_2)$: **return** `BDD_imp`(`BDD_form`$(\varphi_1, \mathcal{I})$, `BDD_form`$(\varphi_2, \mathcal{I})$);
    $\exists x\ \varphi$: **return** `BDD_exists`$(x,$ `BDD_form`$(\varphi, \mathcal{I}))$;
    $\rho x_1 ... x_n$: **return** `BDD_term`$(\rho, \mathcal{I})\{v_1 := x_1\}...\{v_n := x_n\}$;

function `BDD_term` (RelationalTerm ρ, Interpretation $\mathcal{I}$) : Bdd =
  /* Calculates the BDD of term ρ in the interpretation $\mathcal{I}$ */
  case ρ **of**
    $R \in \mathcal{R}$: **return** $\mathcal{I}(R)$ /* pointer to BDD for R */;
    $X \in \mathcal{V}$: **return** X /* name of X */;
    $\lambda x_1 ... x_n\, \varphi$: **return** `BDD_form`$(\varphi, \mathcal{I})\{x_1 := v_1\}...\{x_n := v_n\}$;
    $\mu X\, \rho$: $r :=$ `BDD_term`$(\rho, \mathcal{I})$; **return** `BDD_lfp`$(r, 0)$;

function `BDD_lfp` (BDD r, BDD X^i) : BDD =
  /* Fixpoint iteration of BDD r for ρ with substitution $\{X := X^i\}$ */
  $X^{i+1} := r\{X := X^i\}$;
  if $X^{i+1} = X^i$ **then return** X^i
  else return `BDD_lfp`(r, X^{i+1});

Figure 34: Symbolic evaluation of formulas and terms

coding is generated automatically. The model is given in a C-like input language. It is compiled automatically into an internal BDD representation. Since μcke uses several sophisticated heuristics for the allocation of BDD variables, its performance is comparable to more specialized systems like SMV.

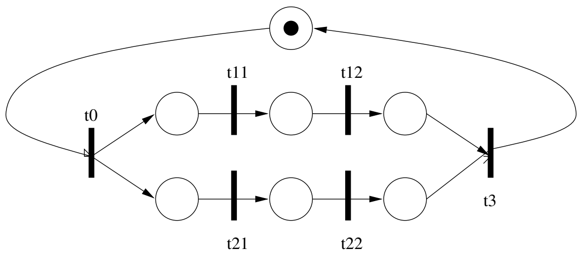
11. Partial Order Techniques

With symbolic methods we try to tackle the complexity problem which arises from the parallel composition of modules by using the BDD data structure which can handle very large sets. Partial order methods, on the other hand, try to avoid the generation of large sets: they only generate a minimal part of the state space which is necessary to evaluate the given formula.

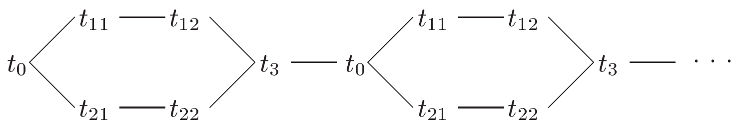
Several variants have been suggested: *stubborn sets* ([Valmari 1990]), *sleep sets* ([Godefroid 1990, Godefroid and Wolper 1991, Godefroid and Pirottin 1993]), *interleaving* and *ample sets* [Katz and Peled 1988, Peled 1993], and others. Subsequently, we describe an algorithm for partial order model checking of linear time temporal logic properties which is based on [Yoneda, Nakade and Tohma 1989, Valmari 1990]. For an overview of other methods, see [Clarke, Grumberg, Minea and Peled 1999]. Partial order methods for branching time logics and symbolic methods have been investigated in [Gerth, Kuiper, Peled and Penczek 1995, Alur, Brayton, Henzinger, Quadeer and Rajamani 1997]. A somewhat different approach to partial order model checking by unfolding of Petri nets was suggested in [McMillan 1992, Esparza 1994].

The interleaving definition of parallel program semantics determines the state space of the global system to be the product of all state spaces of its parallel components. This can lead to wasteful algorithms. In general, each (nondeterministic) execution of a program generates a partial order, where points are ordered by causality. In interleaving semantics this partial order is represented by the set of all of its interleaving sequences.

For example, the following elementary Petri net represents a system with two processes synchronizing via t_0 and t_3:



This system generates the following partial order:



Some of the interleaving sequences are

$t_0\ t_{11}\ t_{12}\ t_{21}\ t_{22}\ t_3\ \ldots$
$t_0\ t_{11}\ t_{21}\ t_{12}\ t_{22}\ t_3\ \ldots$
$t_0\ t_{11}\ t_{21}\ t_{22}\ t_{12}\ t_3\ \ldots$
$t_0\ t_{21}\ t_{11}\ t_{22}\ t_{12}\ t_3\ \ldots$
$t_0\ t_{21}\ t_{11}\ t_{12}\ t_{22}\ t_3\ \ldots$

However, it may not be necessary to consider *all* of these interleavings to determine, e.g., the truth value of the formula $\mathbf{G}^{+}\,\mathbf{F}^{*}\,t_3$. The main idea of partial order methods is to try to inspect only some "representative" interleaving sequences for the formula in question. Thus, we do *not* alter the semantics to deal with "real" concurrency (where independent transitions can occur at the same time), and we do *not* extend the logic to be able to express partial order properties. On the contrary, we will limit the expressiveness of temporal logic and use the partial order to improve the efficiency of model checking.

11.1. Stuttering Invariance

Given an elementary Petri net N and a formula φ, we want to find whether there exists a run ρ of N satisfying φ. In general, there are infinitely many runs through the system; therefore we partition them into a finite number of equivalence classes, such that the existence of a satisfying run ρ implies that every element of the equivalence class $[\rho]$ satisfies φ. Thus we only have to check a finite number of equivalence classes, and a coarser partition yields a better algorithm.

To do so, we need a *stuttering invariant* temporal logic. Consider a formula with the atomic propositions $\{\mathtt{p}_1, ..., \mathtt{p}_k\} \subseteq \mathcal{P}$. Two natural models $\mathcal{M}$ and $\mathcal{M}'$ are *strongly equivalent* with respect to $\{\mathtt{p}_1, ..., \mathtt{p}_k\}$, if they are of the same cardinality, and for all $i \geq 0$ and all $\mathtt{p} \in \{\mathtt{p}_1, ..., \mathtt{p}_k\}$ we have $w_i \in \mathcal{I}(\mathtt{p})$ iff $w_i' \in \mathcal{I}'(\mathtt{p})$. A point w_{i+1} in $\mathcal{M}$ is *stuttering* w.r.t. $\{\mathtt{p}_1, ..., \mathtt{p}_k\}$, if for all $\mathtt{p} \in \{\mathtt{p}_1, ..., \mathtt{p}_k\}$ we have $w_i \in \mathcal{I}(\mathtt{p})$ iff $w_{i+1} \in \mathcal{I}(\mathtt{p})$. For any model $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$, define the *stutter-free kernel* $\mathcal{M}^o$ w.r.t. $\{\mathtt{p}_1, ..., \mathtt{p}_k\}$ to be the model obtained by eliminating all stuttering states from $\mathcal{M}$. More formally, $\mathcal{M}^o$ contains all non-stuttering points from $\mathcal{M}$, and $w \prec w'$ in $\mathcal{M}^o$ iff $w \prec w'$ in $\mathcal{M}$, or there are stuttering points w_1, ..., w_n such that $w \prec w_1 \prec \cdots \prec w_n \prec w'$ in $\mathcal{M}$. Two models $\mathcal{M}_1$ and $\mathcal{M}_2$ are *stuttering equivalent* w.r.t. $\{\mathtt{p}_1, ..., \mathtt{p}_k\}$, if their stutter-free kernels are strongly equivalent w.r.t. $\{\mathtt{p}_1, ..., \mathtt{p}_k\}$.

A formula φ is *stuttering invariant* or *preserved under stuttering*, if for any two models $\mathcal{M}_1$ and $\mathcal{M}_2$ which are stuttering equivalent with respect to the set of atomic propositions of φ it holds that $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_2 \models \varphi$. A language is stuttering invariant, if all of its formulas are stuttering invariant.

In general, formulas involving the operator $\mathbf{X}$ are not stuttering invariant. For example, the formula $\mathbf{X}\,\mathtt{p}$ holds in the model $(\{w_0, w_1, w_2\}, \mathcal{I}, w_0)$, where $\mathcal{I}(\mathtt{p}) = \{w_0, w_1\}$ and $w_0 \prec w_1 \prec w_2$, but not in the stuttering equivalent model $(\{w_0, w_2\}, \mathcal{I}, w_0)$. The next-operator has always been a topic of discussions in temporal specification [Lamport 1983]. Most notions of refinement of systems do not preserve properties with next-operators. Recall that $\mathbf{X}$ is definable with $\mathbf{U}^{+}$, but not with $\mathbf{U}^{*}$ (see Lemma 2.1 and Page 1647). Let $\mathbf{LTL} - \mathbf{X}$ be the logic built from propositions $\mathtt{p} \in \mathcal{P}$, boolean connectives $\perp$, $\rightarrow$ and the reflexive until operator $\mathbf{U}^{*}$.

11.1. Lemma. *Any* $\mathbf{LTL} - \mathbf{X}$ *formula is stuttering invariant.*

Proof: Assume that φ is an $\mathbf{LTL} - \mathbf{X}$ formula, $\mathcal{M} \triangleq (U, \mathcal{I}, w_0)$ a model and $\mathcal{M}^o \triangleq (U^o, \mathcal{I}^o, w_0)$ the stuttering-free kernel of φ w.r.t. the propositions in φ. Furthermore, for any $w \in U$, let $w^o \in U^o$ be the maximal non-stuttering point such that $w^o \leq w$. We show that for any $w \in U$

$$(*) \qquad (U, \mathcal{I}, w) \models \varphi \quad \text{iff} \quad (U^o, \mathcal{I}^o, w^o) \models \varphi.$$

In particular, since $w_0^o = w_0$, this implies that $\mathcal{M} \models \varphi$ iff $\mathcal{M}^o \models \varphi$. From this, the claim follows immediately: if $\mathcal{M}_1^o$ and $\mathcal{M}_2^o$ are strongly equivalent w.r.t. the atomic propositions of φ, then clearly $\mathcal{M}_1^o \models \varphi$ iff $\mathcal{M}_2^o \models \varphi$. If $\mathcal{M}_1$ and $\mathcal{M}_2$ are stuttering equivalent, then the stutter-free kernels $\mathcal{M}_1^o$ and $\mathcal{M}_2^o$ are strongly equivalent. Therefore, in this case $\mathcal{M}_1 \models \varphi$ iff $\mathcal{M}_1^o \models \varphi$ iff $\mathcal{M}_2^o \models \varphi$ iff $\mathcal{M}_2 \models \varphi$.

The proof of $(*)$ is by induction φ. For atomic propositions, $w_{i+1} \models \mathtt{p}$ iff $w_i \models \mathtt{p}$ for each point w_{i+1} which is stuttering w.r.t. $\{\mathtt{p}, \mathtt{p}_1, \ldots, \mathtt{p}_k\}$. Therefore $w \models \mathtt{p}$ iff $w^o \models \mathtt{p}$. For boolean connectives the statement is obvious. For the $\mathbf{U}^{*}$-operator, we treat only the case $\varphi = \mathbf{F}^{*}\psi = (\top\,\mathbf{U}^{*}\,\psi)$; the general case $\varphi = (\psi_2\,\mathbf{U}^{*}\,\psi_1)$ is similar. $(U, \mathcal{I}, w_0) \models \mathbf{F}^{*}\psi$ means that there is a $w_1 \geq w_0$ such that $(U, \mathcal{I}, w_1) \models \psi$. By the inductive hypothesis, this is equivalent to the claim that for some $w_1 \geq w_0$, $(U^o, \mathcal{I}^o, w_1^o) \models \psi$. This claim in turn holds iff for some $v_1 \in U^o$, $v_1 \geq w_0^o$ and $(U^o, \mathcal{I}^o, v_1) \models \psi_1$. This means that $(U^o, \mathcal{I}^o, w_0^o) \models \mathbf{F}^{*}\psi$. Note that this proof is not valid for the $\mathbf{F}^{+}$-operator, since it is possible that $w_1 > w_0$ but $w_1^o = w_0$. $\square$

In [Peled and Wilke 1997], a converse to this lemma is proved:

11.2. Theorem. *Any* $\mathbf{LTL}$ *formula which is stuttering invariant is expressible in* $\mathbf{LTL} - \mathbf{X}$.

Stuttering invariance allows to group all stuttering equivalent runs into the same equivalence class, thereby reducing the average complexity of the model checking. Of course, the reduction will be better if φ uses fewer propositions. Usually, a given formula mentions only a small subset of the system, allowing the equivalence classes to be rather large. In particular, consider a system with two independent transitions t_1 and t_2 (a formal criterion of independence is given below). All runs which differ only in the interleaving of t_1 and t_2 are stuttering equivalent with respect to all atomic propositions not related to t_1 or t_2. Therefore, each $\mathbf{LTL} - \mathbf{X}$ formula not referring to t_1 and t_2 has the same truth value for all of these runs.

11.2. Partial Order Analysis of Elementary Nets

First, we need an appropriate stuttering-invariant restricted logical language to express "interesting" properties of elementary Petri nets. Recall that a state of the net is just a marking of its places. Thus, it is reasonable to use places as atomic propositions, where a proposition p is valid in a state iff the place p is marked in that marking.

Assume that we are given an elementary Petri net and an **LTL** $-$ **X** formula describing a property of this net. Now, we define when two transitions are independent of one another. Firstly, independent transitions must neither disable nor enable each other; that is, if t_1 is enabled in s and s' is a successor of s with respect to the firing of t_1, then t_2 is enabled at s iff t_2 is enabled at s', and vice versa for t_2 firing. Secondly, if the independent transitions t_1 and t_2 are both enabled in s, then they must be able to commute; that is, each execution obtained by first firing t_1 and then t_2 must be stuttering equivalent (w.r.t. the property under consideration) to one obtained by first firing t_2 and then t_1.

However, it is not practical to check these two properties for all pairs of transitions in all global states of the system. Therefore, we use a syntactic condition which ensures that some transition is independent from another one.

Call a set T of transitions *persistent* in s, if whatever one does from s while remaining outside of T does not affect T. Formally, T is persistent in s iff for all $t \in T$ and all firing sequences $t_0, t_1, ..., t_n, t$ such that $t_i \notin T$ for all $0 \leq i \leq n$ there exists a stuttering equivalent firing sequence starting with t.

If T is persistent, we do not have to consider the firing of transitions outside of T when constructing the children of the given state in the depth-first-search; there will be a stuttering equivalent sequence constructed by the firing of some $t \in T$.

However, this definition still is not effective. There is no efficient way to compute a minimal persistent set of transitions for a given state. Therefore, we compute an approximation. There is a tradeoff between the amount of time spent in the calculation of minimal persistent sets, and the reduction of the state space obtained. As a general strategy, some simple heuristics can gain a lot, and sophisticated methods don't add too much.

We start with a single enabled transition $T = \{t\}$ and repeat until stabilization to add all transitions which can "interfere" with some transition in T. Here "interfere" means that they can enable or disable, or cannot commute with some transition in T.

Given any marking m, firable transition t_f and disabled transition t, we have to find a set of firable transitions such that the firing of any transition in this set could lead to the firing of t before t_f. A set $NEC(t, m)$ of transitions is *necessary* for t in m, if $NEC(t, m) = \{t' \mid p \in t'\bullet\}$ for some $p \in (\bullet t \setminus m)$. We use a functional notation here, since $NEC(t, m)$ is determined by the chosen heuristic strategy. Similarly, the set $NEC^*(t, m)$ is defined to be any set of transitions containing t which is transitively closed under necessity; that is, for any $t' \in NEC^*(t, m)$ such that t is disabled in m there exists a set $NEC(t', m)$ of transitions necessary for t' such that $NEC(t', m) \subseteq NEC^*(t, m)$. If t is disabled in m, then t cannot fire unless some transitions from $NEC^*(t,m)$ fire before.

If t is in conflict with t_f, then the firing of any transition in $NEC^*(t,m)$ could eventually enable t; therefore all transitions in $NEC^*(t,m)$ have to be fired as alternatives to the firing of t_f. But, there is still another class of dependent transitions. We want to obtain stuttering equivalence with respect to the atomic propositions in φ. Therefore, we have to take into account that φ might fix an order onto the firing of independent transitions. Usually, φ contains only a few propositions. Call a transition *visible* for φ, if $\bullet t \cup t\bullet$ contains any place p appearing in φ. If t is visible, the firing order with other visible transitions is important. A visible transition can be regarded to be in conflict with all other visible transitions. Define the *conflict* of t by

$$C(t) = \{t' \mid \bullet t' \cap \bullet t \neq \{\}\} \cup \{t\}.$$

The *extended conflict* of t is just the conflict of t, if t is invisible; otherwise, it is the conflict of t plus all other visible transitions. Now a *dependent set* $DEP(t_f,m)$ of t_f is any set of transitions such for any t in the extended conflict of t_f there exists a set $NEC(t,m) \subseteq DEP(t_f,m)$.

Finally, the set of transitions which are fired should be transitively closed under dependency; thus, let $READY(m)$ be any (smallest) nonempty set of firable transitions, such that

$$DEP(t_f,m) \subseteq READY(m) \;\text{ if }\; t_f \in READY(m).$$

Correctness of this reduction method is guaranteed by the following theorem:

11.3. THEOREM. *For any firing sequence ρ of the net there exists a firing sequence ρ'* generated only by firing ready transitions *such that ρ and ρ' are equivalent with respect to all* **LTL − X** *safety properties.*

Consider the depth-first model checking algorithm for **LTL** in Figure 19. During the construction of the set of children of a state in the depth first search we can neglect all firable transitions which are not ready. This can result in a considerable average case reduction; in fact, for examples with many concurrent and "almost" independent processes it can logarithmically reduce the state space which has to be traversed. Though the worst case complexity of constructing a ready set is cubic in the size of the net, in average examples it is only linear in the number of transitions.

The above construction can be extended to deal also with liveness and other linear temporal logic properties. To do so, we need to assure that whenever a state is reached for the second time, a different ready set is constructed, to make sure that no eventuality is delayed infinitely often. For a detailed exposition and an extension to real-time logics, see [Yoneda and Schlingloff 1997].

12. Bounded Model Checking

The model checking algorithms of the previous sections were based on the idea of calculating the greatest or least fixed point of a certain continuous function. Model checking can also be done by translating temporal logic into classical logic and using well-established automated deduction methods. In particular, in Subsection 2.3 on Page 1649 we defined a translation **FOL** from linear temporal logic to first order logic. If the model to be checked is finite, then each first order existential quantifier can be replaced by a finite disjunction, and every universal quantifier can be replaced by a finite conjunction of variables. Moreover, as described in section 10, each finite model can be coded as a boolean combination of atomic formulas $\mathtt{p}(t)$ and $t \prec t'$. Likewise, for sequence-validity, the condition that a finite set $\{t_1, \dots t_n\}$ of points forms a maximal path in a model can be coded as such a formula.

Consider the conjunction of the propositional translation of the formula and the boolean encoding of the model. This is a formula which can be tested for satisfiability by standard SAT algorithms. In [Biere, Cimatti, Fujita and Zhu 1999, Biere, Cimatti and Zhu 1999], the term *bounded model checking* is introduced for checking sequence-validity of future **LTL** formulas with this approach. The execution sequences of a Kripke model are enumerated by increasing length and combined with the translation of the formula. These are converted into conjunctive normal form and tested for satisfiability by propositional theorem provers. With appropriate heuristics, in some cases this method turned out to give even better results than BDD based methods.

12.1. An Example

Before giving the technical details, we show an example. Consider the Kripke model in Fig. 35. There are four points in the model. Each point w is represented by two

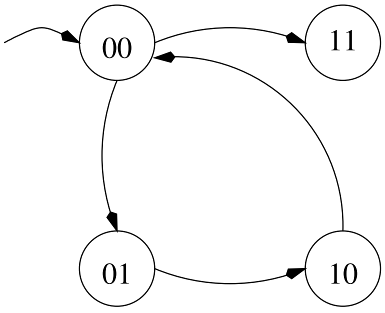


Figure 35: A two-bit model

state variables, $w \triangleq (v_1, v_0)$, denoting the value of the high bit and the low bit, respectively. The initial state is (00). Thus the initial state predicate $I(w)$ is defined as $(\neg v_1 \wedge \neg v_0)$. The only terminal state is (11), thus the terminal state predicate $T(w)$ is $(v_1 \wedge v_0)$. The transition relation is represented by the formula $R(w, w') \triangleq (\neg v_1 \wedge \neg v_0 \wedge \neg v_1' \wedge v_0') \vee (\neg v_1 \wedge v_0 \wedge v_1' \wedge \neg v_0') \vee (v_1 \wedge \neg v_0 \wedge \neg v_1' \wedge \neg v_0') \vee (\neg v_1 \wedge \neg v_0 \wedge v_1' \wedge v_0')$

Suppose we are interested in the fact that any execution eventually reaches state (11). In **LTL**, this amounts to checking whether $\mathbf{F}^{*}(v_1 \wedge v_0)$ is sequence-valid. Equivalently, we can check whether there is a maximal path in the model in which state (11) is never reached. That is, we check whether $\mathbf{G}^{*}\neg(v_1 \wedge v_0)$ is satisfiable in the model. According to the definition, this is the case iff there is a path in the model starting in an initial point and ending in a terminal point or in a cycle, such that every point on the path satisfies $\neg v_1$ or $\neg v_0$. In bounded model checking, we restrict our attention to paths of length k, that is, paths with $k+1$ states. We start with $k = 0$, and increment k until a witness is found. Consider the case where k equals 2. We name the $k+1$ states as w^0, w^1, w^2. Since every state is encoded by two boolean variables, there are six propositional variables altogether: v_1^0, v_0^0, v_1^1, v_0^1, v_1^2, v_0^2. We now formulate a set of constraints on these variables in propositional logic which guarantee that the path $\sigma = (w^0, w^1, w^2)$ is indeed a witness for $\mathbf{G}^{*}(\neg v_1 \vee \neg v_0)$.

- First, σ must start in an initial point. This is expressed by $I(w^0)$ as described above: $\varphi_1 \triangleq (\neg v_1^0 \wedge \neg v_0^0)$
- Second, each w^{i+1} must be a successor of w^i according to the transition relation, i.e., $R(w^0, w^1) \wedge R(w^1, w^2)$ must hold. This expands to
$$\begin{aligned}\varphi_2 \triangleq\, &(\neg v_1^0 \wedge \neg v_0^0 \wedge \neg v_1^1 \wedge v_0^1) \vee (\neg v_1^0 \wedge v_0^0 \wedge v_1^1 \wedge \neg v_0^1) \vee\\ &\quad (v_1^0 \wedge \neg v_0^0 \wedge \neg v_1^1 \wedge \neg v_0^1) \vee (\neg v_1^0 \wedge \neg v_0^0 \wedge v_1^1 \wedge v_0^1) \wedge\\ &(\neg v_1^1 \wedge \neg v_0^1 \wedge \neg v_1^2 \wedge v_0^2) \vee (\neg v_1^1 \wedge v_0^1 \wedge v_1^2 \wedge \neg v_0^2) \vee\\ &\quad (v_1^1 \wedge \neg v_0^1 \wedge \neg v_1^2 \wedge \neg v_0^2) \vee (\neg v_1^1 \wedge \neg v_0^1 \wedge v_1^2 \wedge v_0^2)\end{aligned}$$
- Third, the path must be either terminal or end in a loop. That is, either $T(w^2)$ holds, or there must be a transition from w^2 to one of w^0, w^1 or w^2. The formula $\varphi_3 \triangleq T(w^2) \vee R(w^2, w^0) \vee R(w^2, w^1) \vee R(w^2, w^2)$ is expanded similar to φ_2.
- Fourth, $\mathbf{G}^{*}\neg(v_1 \wedge v_0)$ must hold in the first point of the sequence, i.e., $\neg(v_1 \wedge v_0)$ must hold for w^0, w^1 and w^2. Therefore, $\varphi_4 \triangleq \bigwedge_{i=0}^{2} \neg(v_1^i \wedge v_0^i)$.

It is easy to see that there is a propositional model for $\varphi \triangleq \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ iff there is a maximal path consisting of three model states validating the given formula. Satisfiability of φ can be checked by SAT procedures like SATO [Zhang 1997] or Stålmarck's algorithm [Stålmarck 1989, Stålmarck and Säflund 1990, Borälv 1997]. Thus, by increasing the number of states allowed in the search, we get an alternative model checking procedure.

In this example, the formula is indeed satisfiable. The satisfying assignment corresponds to a counterexample that is a path from the initial point (00) over (01) to (10) followed by the loop from (10) to (00). If the transition from (10) to (00) is changed to point (11), then the original formula becomes unsatisfiable.

12.2. Translation into Propositional Logic

Assume that we are given a Kripke model $\mathcal{M}$, an **LTL** formula ψ and a bound k. Subsequently, each w^i is a vector of $\lceil \log |\mathcal{M}| \rceil$ boolean variables. We will construct a propositional formula in $w^0 \dots w^k$ which is (propositionally) satisfiable iff there is a maximal path of length k in $\mathcal{M}$ validating ψ.

The initial and terminal state predicates $I(w)$ and $T(w)$ and the transition relation $R(w, w')$ are given by $\mathcal{M}$. The following propositional formula describes that the points to which the variables $w^1 \dots w^k$ refer form a maximal path in $\mathcal{M}$:

$$[\![\mathcal{M}]\!]_k \triangleq I(w^0) \wedge \bigvee_{i=1}^{k} R(w^{i-1}, w^i) \wedge \Big(T(w^k) \vee \bigvee_{l=0}^{k} R(w^k, w^l)\Big)$$

Now we define the translation $[\![\psi]\!]_k^i$ of a temporal formula ψ evaluated at point w^i in the sequence $(w^0 \dots w^k)$. In general, the constraint imposed by the temporal specification depends on whether the path under consideration is terminating or not. Consider the formula $(\varphi\, \mathbf{U}^{+}\, \psi)$ in a terminating path $(w^0 \dots w^k)$. This formula holds in point w^i iff there is a $i < j \leq k$ such that such that ψ holds at w^j, and φ holds at all w^m such that $i < m < j$. This can be translated by a disjunction over all possible positions w^j at which ψ eventually might hold, and a conjunction for each of these positions ensuring that φ holds for all points between w^i and w^j. That is, in this case $[\![(\varphi\, \mathbf{U}^{+}\, \psi)]\!]_k^i \triangleq \bigvee_{j=i+1}^{k}([\![\psi]\!]_k^j \wedge \bigwedge_{m=i+1}^{j-1}[\![\varphi]\!]_k^m)$

Now consider the case that the path $(w^0 \dots w^l \dots w^k)$ ends with a loop from w^k to w^l. The formula $(\varphi\, \mathbf{U}^{+}\, \psi)$ is satisfied in w^i iff one of the following holds:

- as for terminating sequences, there exists some $i < j \leq k$ such that ψ holds at w^j, and φ holds at all w^m such that $i < m < j$, or
- there exists some $l \leq j \leq i$ such that ψ holds at w^j, and φ holds at all w^m such that $i < m \leq k$, and φ holds at all w^m such that $l \leq m < j$.

Figure 36 visualizes these two possibilities.

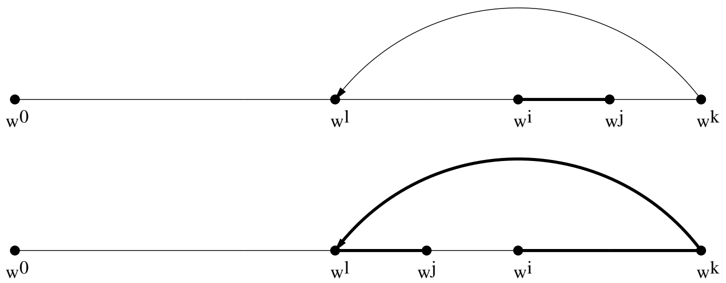


Figure 36: Two possibilities for "until" in a loop

The definition of $[\![\psi]\!]_k^i$ is by recursion on the structure of ψ, where the current point i changes but the length of the path k stays the same. For this translation, let $i \leq k$ be natural numbers, and let $(\bigvee_{j=l}^{i}\psi) \triangleq \bot$ for $l > i$.

- $[\![\mathtt{p}]\!]_k^i \triangleq \mathtt{p}(w^i)$
- $[\![\bot]\!]_k^i \triangleq \bot$
- $[\![(\varphi \rightarrow \psi)]\!]_k^i \triangleq ([\![\varphi]\!]_k^i \rightarrow [\![\psi]\!]_k^i)$

- $[\![(\varphi\, \mathbf{U}^{+}\, \psi)]\!]_k^i \triangleq \bigvee_{j=i+1}^{k}([\![\psi]\!]_k^j \wedge \bigwedge_{m=i+1}^{j-1}[\![\varphi]\!]_k^m) \vee$
$\bigvee_{l=0}^{k}\Big(\bigwedge_{m=i+1}^{k}[\![\varphi]\!]_k^m \wedge R(w^k, w^l) \wedge \bigvee_{j=l}^{i}\big([\![\psi]\!]_k^j \wedge \bigwedge_{m=l}^{j-1}[\![\varphi]\!]_k^m\big)\Big)$

For the last of these clauses, cf. Figure 36. Correctness of our translation can be stated as follows.

12.1. Theorem. *There exists a maximal path of length k generated by $\mathcal{M}$ which initially validates ψ iff $([\![\mathcal{M}]\!]_k \wedge [\![\psi]\!]_k^0)$ is propositionally satisfiable. In other words, ψ is sequence-valid in $\mathcal{M}$ iff $([\![\mathcal{M}]\!]_k \to [\![\psi]\!]_k^0)$ is propositionally valid for all $k \geq 0$.*

An upper bound for the length k of the path to be considered is $|\mathcal{M}| \times 2^{|\psi|}$ (for the complexity of **LTL** model checking, see Sect. 8.2). In principle, bounded model checking could be extended to other specification logics such as μ**TL**. In practice, however, the number of boolean propositions which are introduced tends to be too big for currently available SAT provers.

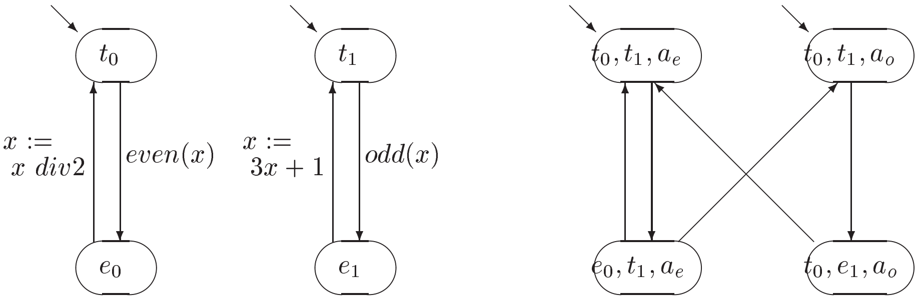
13. Abstractions

Even though BDD representations, partial order methods and SAT procedures allow to apply model checking to rather large systems, one of the main topics still is the size of the models. To verify an implementation of several thousands of lines of code by model checking, it is necessary to find a suitable abstraction.

13.1. Abstraction functions

Numerous authors have considered the problem of reducing the complexity of verification by using abstractions, equivalences, preorders, etc. For example, in [Graf and Steffen 1990] a method is described for generating a reduced version of the global state space, given a description of how the system is structured and specifications of how the components interact. In [Wolper 1986] it is demonstrated how to do model checking for programs which are data independent. The method described in [Kurshan 1989], which is based on ω-language containment, was implemented in the COSPAN system [Har'El and Kurshan 1990, Kurshan 1994]. In this system, the user may give abstract models of the system and specification in order to reduce the complexity of the test for containment. To ensure soundness, the user specifies homomorphisms between actual and abstract processes. These homomorphisms are checked automatically. We describe a general framework elaborated in [Long 1993].

Traditionally, finite-state verification methods focus on the control flow of the system. Symbolic methods have made it possible to handle even some systems that involve nontrivial data manipulation, but the complexity of verification is often high. However, specifications of systems that include data paths usually involve fairly simple relationships among the data values in the system. For example, the correctness of a communications protocol might be independent of the particular data transmitted, provided that no two subsequent messages are identical. As another example, in verifying the addition operation of a microprocessor, we might require that the value in one register is eventually equal to the sum of the values in two other registers. The complexity of the verification can be reduced in such cases by suitable abstractions. An *abstraction* is specified by giving a mapping between the actual data values in the system and a small set of *abstract data values*. By extending the mapping to states and transitions, we can produce an abstract version of the system under consideration. The abstract system is often much smaller than the actual system, and, as a result, it is usually much simpler to verify properties at the abstract level.



(a) Concrete system (b) Abstraction

Figure 37: The dining mathematicians

For example, consider the program from Figure 37. This example called the "dining mathematicians" is from [Dams et al. 1994] and is reconsidered in [Merz 1997]. It consists of two processes communicating via a shared variable x which ranges over the domain D_x of all integers. Initially x is any positive integer. Both processes have a "thinking" and an "eating" state and start in the former. That is, the state space is $\{t_0, e_0\} \times \{t_1, e_1\} \times D_x$, and the initial states are $\{(t_0, t_1, d) \mid d > 0\}$. Note that both of these sets are infinite. The system ensures mutual exclusion to the eating phase and starvation-freeness for both processes.

Assume that we are interested in proving mutual exclusion: $\mathbf{A}\,\mathbf{G}^{*}\,\neg(e_0 \wedge e_1)$. We create a domain A_x of abstract values for x, with $A_x \triangleq \{a_z, a_e, a_o\}$, and define the *abstraction mapping* h_x from D_x to A_x as follows.

$$\alpha_x(d) \triangleq \begin{cases} a_z, & \text{if } d = 0, \\ a_e, & \text{if } d \text{ is even, and} \\ a_o, & \text{if } d \text{ is odd.} \end{cases}$$

Now we can use just three atomic propositions to express the abstract value of x: "$x \mathrel{\widehat{=}} a_0$", "$x \mathrel{\widehat{=}} a_e$", and "$x \mathrel{\widehat{=}} a_o$". We can no longer express properties about the exact value of x using these atomic propositions. In many cases though, by judicious choice of the abstraction mapping, knowing just the abstract value is sufficient.

Two points $w_0 = (w_{00}, w_{01}, d_0)$ and $w_1 = (w_{10}, w_{11}, d_1)$ in the original Kripke model are *equivalent w.r.t.* the abstraction mapping α, if $w_{00} = w_{10}$, $w_{01} = w_{11}$ and $\alpha_x(d_1) = \alpha_x(d_2)$. That is, two points are equivalent if they have the same label, and the abstracted variable values in both points are equal. The *α-abstraction* is the quotient of the original model under this equivalence. Since the abstract domain A_x is finite, the α-abstraction is a finite Kripke model. Figure 37(b) shows the reachable part of the α-abstraction of 37(a). It is easy to see (and can be confirmed by model checking) that the abstracted system validates $\mathbf{A}\,\mathbf{G}^{*}\neg(e_0 \wedge e_1)$. As we will see below, this implies that the original systems also guarantees this property.

Formally, abstractions are formed by giving surjections $\alpha_1, \ldots, \alpha_n$ which map each D_i onto a set D_i^{α} of abstract values. The surjection $\alpha = (\alpha_1, \ldots, \alpha_n)$ then maps each program state to a corresponding abstract state. As explained above, this mapping may be applied in a natural way to the initial states and the transitions of the program. The resulting transition system is the α-abstraction of the original program. Applying abstractions to several or all of the program variables, the specification has a much smaller number of atomic propositions and points. For the abstracted system, various state space reductions discussed in previous sections can be applied.

One way of obtaining a representation of the α-abstraction of a concurrent program is to build a representation of the original state space and to construct the α-abstraction from it. However, if the original state space is infinite as in the above example, or it is too large to fit into memory, this may not be feasible. In the finite state case, it might be possible to represent the system using BDD-based methods, but the computational complexity of building the α-quotient from this representation can still be very high.

To circumvent these problems, another way of producing abstract models in a BDD-based verification tool is to start with a high level description of the system and the abstraction function. The system could be given, e.g., as a program in a hardware description language. From this, a BDD for the abstracted system is generated directly. In order to perform the compilation process effectively, an *approximation* to the α-abstraction is generated[Clarke, Grumberg and Long 1994*a*]. This approximation might be somewhat larger than the α-abstraction, but it can be built very efficiently. The techniques used in this construction are similar to those involved in *abstract interpretation* [Cousot and Cousot 1977, Cousot and Cousot 1979, Dams 1995]. This way, it is even possible to use abstractions to verify systems in which the data path is not completely specified. By modeling the data path as a collection of units that perform unspecified functions, the verification of the data path and the verification of the control can be largely decoupled.

To be able to interpret specification formulas with respect to both the original transition system and its abstraction, atomic formulas must be those specifying that a program variable has a particular abstract value. In Theorem 4.11 we showed that if $\mathcal{M}_1$ is simulated by $\mathcal{M}_2$, then any formula in the logic **ACTL** valid in $\mathcal{M}_2$ is also valid in $\mathcal{M}_1$. An abstraction is a special simulation; thus if an **ACTL** formula is true in the abstract system, we can conclude that it is also true in the original system. In addition, if the equivalence relations induced by the α_i are *congruences* with respect to the operations used in the program, then the formula is true in the abstract system iff it is true of the original system. [Loiseaux, Graf, Sifakis, Bouajjani and Bensalem 1995] discusses abstraction techniques which preserve properties specified in $\mu\mathbf{TL}$.

It should be emphasized that the choice of suitable abstractions α_i is an interactive step in the verification. Usually, there are several possibilities to abstract a given system, all preserving different properties. In our above example, the chosen abstraction does not allow to prove starvation-freeness of the second process. However, this situation is not typical for industrial applications. In [Clarke, Grumberg and Long 1994*a*], the following abstractions are used to verify a pipelined arithmetic/logical unit with over 4000 state bits and 10^{1300} reachable states.

- congruence modulo an integer, for dealing with arithmetic operations;
- single bit abstractions, for dealing with bitwise logical operations;
- product abstractions, for computing abstractions such as the above; and
- symbolic abstractions. This is a powerful type of abstraction that allows to verify an entire class of formulas simultaneously.

Another approach at implementing abstraction functions is directly at the level of the BDD data structure. Given an abstraction function, we can reduce the size of a BDD by merging nodes that have the same abstract value. *Abstract BDDs* (ABDDs) are a generalization of Residue BDDs (RBDDs, see [Kimura 1995]). To obtain an ABDD it is not necessary to build the full BDD: ABDDs can be constructed directly from the abstraction function and the description of the system. For more information, see [Clarke, Jha, Lu and Minea 1997].

13.2. Symmetry Reductions

Most large hardware circuits are highly symmetric. For instance, one can find symmetry in memories, caches, register files, bus and network protocols — any type of hardware containing replicated structures. For symmetric systems, we can apply special abstractions to avoid searching the entire state space of the circuit and to reduce the size of the BDDs representing the transition relation[Starke 1991, Emerson and Sistla 1993, Clarke, Filkorn and Jha 1993, Ip and Dill 1993].

Suppose that we want to represent the boolean function (formula) $\varphi(v_1, ..., v_n)$ of n variables by a BDD. Symmetry in a boolean function is modeled in terms of a permutation group acting on the set of variables of the function. We say that φ is *invariant* under a permutation σ on v_1, ..., v_n, if the value of the function does not change when the permutation σ is applied to its arguments:

$$\varphi(v_1, ..., v_n) = \varphi(\sigma(v_1), ..., \sigma(v_n))$$

The function is said to be invariant under a group G of permutations, if it is invariant under each permutation σ in G. For example, let $\varphi(v_1, v_2, v_3, v_4)$ be the function which tests whether two 2-bit numbers (v_1, v_3) and (v_2, v_4) are equal. The function φ is clearly invariant under the transpositions (1 2) and (3 4). The first permutation corresponds to exchanging input bits v_1 and v_2. The second corresponds to exchanging v_3 and v_4. The function will, of course, also be invariant under the group generated by the two transpositions.

Let B^n be the set of boolean vectors of length n, and let G be a permutation group on $1, ..., n$. Assume that G acts on B^n in the natural way. For example, applying the transposition (2 3) to $(0, 1, 0, 1)$ yields $(0, 0, 1, 1)$. We say that two vectors v^1 and v^2 are *equivalent with respect to* G if there is a permutation σ in G such that $v^1 = v^2$. Since G is a group, this relation is an equivalence relation on B^n and therefore partitions B^n into a number of equivalence classes. The number of equivalence classes may be much smaller than the number of boolean vectors in B^n.

A boolean function $\varphi(v_1, ..., v_n)$ is uniquely determined by the set of vectors in B^n that cause it to have the value $\top$. If φ is invariant under some group of permutations G, it may be possible to compact the BDD representation for φ: if any one of the boolean vectors in some equivalence class determined by G makes φ true, then all of the vectors in this equivalence class will. Consequently, in the BDD representation for φ it is necessary to keep at most one representative from each equivalence class. In many cases this significantly reduces the size of the BDD for φ.

Essentially the same idea can be used to reduce the size of the state space that must be searched by the symbolic model checking algorithm. Let U be the set of possible states of the system, which are determined by the values of v_1, ..., v_n. A permutation of these state variables induces a permutation on the state-space of the system. Let $\prec$ be the transition relation of the system and $\equiv$ be an equivalence relation. We say that $\prec$ *respects* $\equiv$ if whenever $w_1 \equiv w_1'$ and $w_2 \equiv w_2'$, then $w_1 \prec w_2$ iff $w_1' \prec w_2'$. When the transition relation R respects the equivalence relation $\equiv$ determined by a permutation group, it is possible to reduce the state space to the set of equivalence classes $U^{\equiv}$ determined by $\equiv$. The corresponding transition relation between these equivalence classes is $\prec^{\equiv}$. Since we only need one point for each equivalence class, the model $(U^{\equiv}, \prec^{\equiv})$ is often much smaller than the original model $(U, \prec)$.

Similar as with abstraction functions, the reduced BDD can be constructed directly from a description of the system and the permutation group. For more information, the reader is referred to [Kannan and Lipton 1986, Clarke, Filkorn and Jha 1993]. It is not clear, though, how the reductions obtained by symmetries interacts with other abstraction techniques and partial order methods.

13.3. Parameterized Systems

A special case of a symmetry is that the system consists of an arbitrary number of similar or identical processes. Systems of this type are commonplace – they occur in bus protocols and network protocols, I/O channels, and many other structures that are designed to be extensible by adding similar components. A number of methods have been proposed for extending model checking to such designs [Clarke, Grumberg and Browne 1986, Wolper and Lovinfosse 1989, German and Sistla 1992, Clarke, Grumberg and Jha 1995].

After using a model checker to determine the correctness of a system configured with a fixed number of processors or other components, it is natural to ask whether this number is enough in some sense to represent a system with any number of components. This question was approached in [Browne, Clarke and Grumberg 1989], who extended **CTL** to a logic called *indexed* **CTL**. This logic allows the restricted use of process quantifiers as in the formula $\bigwedge_i \varphi_i$, which means that the formula φ holds for all processes i. Restricting the use of these quantifiers and eliminating the next-time operator makes it impossible to write a formula which can distinguish the number of processes in a system. By establishing an appropriate relationship between a system with n processes and a system with $n+1$ processes, one can guarantee that all systems satisfy the same set of formulas in the indexed logic. This method was used to establish the correctness of a mutual exclusion algorithm by exhibiting a bisimulation relation between an n-process system and a 2-process system, and applying model checking to the 2-process system.

One disadvantage of the indexing method is that the bisimulation relation must be proved "by hand" in an *ad hoc* manner. Finite state methods cannot be used to check it because it is a map between states of a finite state process and a process with an arbitrary number of states. A method without this disadvantage was proposed in [Kurshan and McMillan 1989], and independently in [Wolper and Lovinfosse 1989]. This method uses a process Q to act as an invariant, as the number of processes increases. If P represents one process in the system, then by showing that the possible executions of P composed with Q are contained in the possible executions of Q, we can conclude by induction that Q adequately represents a system of any finite number of processes. Since both P composed with Q and Q are finite state processes, the containment relation can be checked automatically. This method has been applied in [McMillan and Schwalbe 1992] to the Encore Gigamax cache consistency protocol. By slightly generalizing the model of one processor, an invariant process for this system could be obtained which stands for any number of processors on a bus.

These induction techniques have been generalized by a number of authors (e.g., [Marelly and Grumberg 1991]). However, the main problem in all of these verification methods is that of constructing the invariant process. Currently, the invariant process must be generated interactively. Counterexamples produced by model checking tools are helpful for guiding the construction, but it would be useful to have automated techniques for this purpose. To make these methods generally accepted, more results on the combination of model checking and inductive theorem proving and powerful heuristics are necessary.

14. Compositionality and Modular Verification

As explained in Section 9, most circuits and protocols are modeled as networks of communicating parallel processes. The complexity of these models grows exponentially in the number of processes; thus, monolithic verification of such designs can be hard. Therefore, it may be necessary to verify small components separately and, from that, derive the correctness of the whole design, without building a model for the entire system. This so-called *compositionality paradigm* has been investigated by a number of authors [deRoever, Langmaack and Pnueli 1998].

14.1. Model Checking and Theorem Proving

Assume a process consisting of a parallel composition of several subprocesses, where all subprocesses have associated formulas specifying their properties. Whenever a property of a parallel composition is to be proven, we can first prove for each component that the corresponding property holds, and then infer in an adequate proof system that the global property of the composition also holds. Model checking can be used to verify the individual components; then theorem proving techniques can be used to derive global properties of their parallel composition. The composition step substantially simplifies the verification problem, since it avoids building the global state space. Thus, the compositionality paradigm is a promising perspective for the combination of model checking with theorem proving.

Moreover, this approach supports the hierarchical design process. One can work out specifications for all parts of a complex system and prove that if every component satisfies its specification, then the whole system is correct. When the system is implemented it is sufficient to verify each component separately. It is also possible to change the actual implementation of some component without having to repeat the verification of the entire system as soon as the new implementation meets its local requirements.

For instance, consider the problem of verifying a communications protocol that is modelled by three processes: a transmitter, some type of network, and a receiver. Suppose that the specification for the system is that data is eventually transmitted correctly from the sender to the receiver. Such a specification might be decomposed into three local properties. First, the data should eventually be transferred correctly from the transmitter to the network. Second, the data should eventually be transferred correctly from one end of the network to the other. Finally, the data should eventually be transferred correctly from the network to the receiver. We might be able to verify the first of these local properties using only the transmitter and the network, the second using only the network, and the third using only the network and the receiver. By decomposing the verification in this way, we never have to compose all of the processes and therefore avoid the state explosion phenomenon.

Whereas model checking for the verification of the individual components is a well-understood technique, for the derivation of global system properties from local components properties an appropriate calculus is needed. There are two possibilities for implementing a proof system for such a calculus. The first is to incorporate the calculus into a general purpose theorem prover. For example, there are embeddings of Lamport's temporal logic of actions (TLA) into the theorem provers LARCH, PVS and Isabelle (see, e.g., [Abadi, Lamport and Merz 1996]). However, the computational complexity inherent in such an approach may prevent the resulting tool from being applicable for large industrial designs.

The second possibility is to build a special purpose theorem prover for the chosen calculus. Several suggestions for a concrete framework following this approach have been made. The proof system of [Stirling 1987] is, probably, the most compositional in the sense that it clearly reduces the verification problem to the verification of components. However, the logic which is used in this paper is too weak to be of much interest in practice. In [Andersen, Stirling and Winskel 1994] the parallel composition operator was eliminated basically by encoding one of the subprocesses into the formula. In the worst case this results in an exponential blow-up in the size of the formula, and the total complexity remains the same as for non-compositional model checking. The proof system of [Dam 1995] is complete for finite-state processes. However, it uses a silent τ action for all synchronizations, and in the τ-rule there have to be as many premises as there are actions in the model. Therefore, one can only have a fixed set of actions.

The STeP system [Bjørner, Browne, Chang, Colón, Kapur, Manna, Simpa and Uribe 1995, Bjørner, Browne, Chang, Colón, Kapur, Manna, Sipma and Uribe 1996] implements another approach to combining model checking and theorem proving under a single framework. However, the user must decide what has to be model checked and what to be derived in a theorem prover. It would be desirable to create the verification agenda automatically, such that the user will only have to supply some intermediate properties and possibly assist the theorem prover during a proof search.

14.2. Compositional Assume-Guarantee Reasoning

Ideally, compositional reasoning exploits the natural decomposition of a complex system into simpler components, handling one component at a time. In practice, however, when a component is verified it may be necessary to *assume* that the environment behaves in a certain manner. If the other components in the system *guarantee* this behavior, then we can conclude that the verified properties are valid in the entire system. These properties can be used to deduce additional global properties of the system.

The *assume-guarantee paradigm* (cf. e.g., [Pnueli 1984]) uses this method. Typically, a formula is a triple $\langle\psi\rangle\mathcal{M}\langle\varphi\rangle$ where ψ and φ are temporal formulas and $\mathcal{M}$ is a program. The formula is valid if whenever $\mathcal{M}$ is part of a system satisfying ψ, the system must also satisfy φ. A typical proof shows that $\langle\psi\rangle\mathcal{M}\langle\varphi\rangle$ and $\langle\top\rangle\mathcal{M}'\langle\psi\rangle$ hold and concludes that $\langle\top\rangle\mathcal{M} \parallel \mathcal{M}'\langle\varphi\rangle$ is valid. This proof strategy can also be expressed as an inference rule:

$$\frac{\begin{array}{c}\langle\psi\rangle\mathcal{M}\langle\varphi\rangle\\ \langle\top\rangle\mathcal{M}'\langle\psi\rangle\end{array}}{\langle\top\rangle\mathcal{M} \parallel \mathcal{M}'\langle\varphi\rangle}$$

The soundness of an assume-guarantee rule of this form is straightforward. A more powerful form that also involves pure temporal reasoning is:

$$\frac{\begin{array}{c}\langle\psi_1\rangle\mathcal{M}_1\langle\varphi_1\rangle\\ \langle\psi_2\rangle\mathcal{M}_2\langle\varphi_2\rangle\\ \xi_1\wedge\varphi_1\rightarrow\psi_2\\ \xi_1\wedge\varphi_2\rightarrow\psi_1\\ \varphi_1\wedge\varphi_2\rightarrow\xi_2\end{array}}{\langle\xi_1\rangle\mathcal{M}_1\parallel\mathcal{M}_2\langle\xi_2\rangle}$$

In the composed system $\mathcal{M}_1 \parallel \mathcal{M}_2$, the module $\mathcal{M}_2$ is part of the environment of $\mathcal{M}_1$ and vice versa. $\mathcal{M}_2$ guarantees via φ_2 that the assumption ψ_1 of $\mathcal{M}_1$ is met, provided that its own assumption ψ_2 holds. $\mathcal{M}_1$, in turn, guarantees the assumption of $\mathcal{M}_2$, provided that its assumption holds.

As shown in [Pnueli 1984], careless application of this rule may lead to a circular reasoning and, thus, may result in an erroneous conclusion. To avoid this, Pnueli suggested associating a parameter over some well–founded set with each temporal formula in the assume-guarantee rule. The rule then allows for a temporal formula to be deduced only from formulas with smaller parameters. For an abstract account of composition, see [Merz 1997]. Several tools have been developed that permit this type of reasoning to be automated [Josko 1993, Long 1993, Grumberg and Long 1994]. The tools provide a machinery for checking automatically the validity of formulas of the form $\langle\psi\rangle\mathcal{M}\langle\varphi\rangle$. These tools, however, suffer from two main deficiencies.

Firstly, they do not provide any mechanism to avoid or to locate circular reasoning. Thus, they count on the user "common sense" for correct application of the method. An open problem is to develop an algorithm for checking non-circularity in assume-guarantee reasoning. This could bridge the gap between the abstract assume-guarantee paradigm and its computerized version.

Secondly, in order to obtain a powerful method, the preorder and the semantics of the logics should both include a notion of *fairness*. This is essential for modelling systems (hardware or communication protocols) at the appropriate level of abstraction. Unfortunately, no efficient technique exists to check or compute *fair preorder* between models. In [Grumberg and Long 1994], it is suggested how to check the fair preorder in some simple cases. In the general case, the problem is PSPACE-hard [Kupferman and Vardi 1996]. A notion of fair preorder that, on the one hand, is suitable for computerized assume-guarantee reasoning, and on the other hand, can be checked efficiently, would make compositional reasoning less error prone and could widen the applicability of this type of reasoning.

15. Further Topics

There are several extensions to each of the topics presented here, and in many areas there is a lot of ongoing activity. Current research can be classified into two main tracks:

- improve efficiency and applicability of present model checking techniques, and

- extend the realm of application and merge model checking with other formal methods.

A number of papers on industrial case studies, advanced heuristics, and improved algorithms and data structures follows the first track. The second track encompasses papers on model checking for infinite state systems, integration with simulation and testing, as well as model checking for real-time, probabilistic and security related applications.

15.1. Combination of Heuristics

Partial order techniques attempt to alleviate the state explosion problem by constructing a reduced state space to be searched by the model checking algorithm. Originally introduced in the context of untimed models, they have been expanded to handle real time systems [Yoneda and Schlingloff 1997, Sloan and Buy 1997]. In turn, symbolic techniques have been applied to model checking for real-time systems. It seems to be a challenging task to combine the advantages of partial order reduction with a symbolic representation for real-time system verification. One of the intrinsic difficulties is that the partial order reduction, as described in section 11, needs to have access to the search history, which is trivially implemented for explicit state search but has no immediate correspondence in the symbolic case. Recent advances [Alur et al. 1997, Kurshan, Levin, Minea, Peled and Yenigun 1997] have shown that this technique can be combined with symbolic model checking, which in many cases allows much larger state spaces to be handled. One of these methods [Kurshan et al. 1997] allows partial order reduction to be performed statically, by analyzing the state graph of each asynchronous system component. Existing partial order methods for real-time models are dynamic in the sense that they use timing information obtained during the state space search. However, probably a significant part of the dependency information can be obtained statically as well, making the combination with symbolic techniques possible [Minea 1999].

Partial order methods mostly have been investigated within the context of (stutter-invariant) linear temporal logic model checking. The method reduces the complex part of the model checking problem, namely the size of the model. The tableau for an **LTL** specification is usually small. However, the state explosion problem is even more prevalent in the case of conformance checking and (bi-)simulation between automata. A problem here is how to apply partial order reduction simultaneously to both models.

A key factor that affects the efficiency of partial order reduction is the number of visible transitions, i.e., transitions that may change a predicate in the checked property. With more and more complex specifications, the number of visible transitions increases and less reduction can be achieved. Some approaches to alleviate this problem have been proposed in [Peled 1993]. One possibility is to take advantage of the structure of the specification and rewrite it as a combination of simpler properties. However, no optimal solution is known to date.

Other issues that are important in conjunction with the combination of heuristics are abstraction and compositionality. It is still unknown how the efficiency improvement gained by symbolic representation and partial order analysis interact with abstraction techniques and compositional reasoning. To be able to verify even bigger systems, it is important to develop methods and tools that allow to combine the benefits of several methods.

15.2. Real Time Systems

Within the last few years, several attempts have been made to apply formal analysis methods also to *real time systems*. The ideas and techniques presented so far are well-suited for the verification of systems in which only *causal* aspects of time are important. In some applications it is desirable to consider *quantitative* aspects of timing behavior. We say that a system has to satisfy *hard real time constraints*, if its correctness depends on the value or progress of "the real" clock. In hard real-time systems, not only the relative order of events is important, but also their absolute duration with respect to a (conceptual) global clock. For example, in a traffic light controller, it might not be sufficient to show that if a pedestrian pushes a button, then *eventually* the green lights will be on. To allow approaching cars to pass, the light should stay red after the button has been pushed *for at least 10 seconds*. To avoid that pedestrians start crossing at red, it should also change *not later than 30 seconds* after the request. In this example, we assume that both the pedestrian and the traffic light controller have the same measure of the duration of a second. Of course, it is possible to model the global clock as separate concurrent part of the system. Then this global clock synchronizes the local clocks of both pedestrian and traffic light controller. Thus, it is possible to consider real-time verification as special case of the untimed methods described above. However, in hard real-time systems, global time is ubiquitous, therefore this approach may not be the most efficient.

It is important to note that "hard real time" does not mean "as fast as possible". As the above example shows, predictability of timing behavior can also mean that some events do not occur before a certain amount of time has elapsed. As another example, consider a real-time protocol, where all necessary computation steps *must* be performed in *exactly* a fixed time slot. Currently, hard real time systems are designed with trial and error: if a component is too fast, an idle waiting loop is incorporated; if it is too slow, more expensive hardware is used. This procedure has several disadvantages. Firstly, it can add intricate hardware-software dependencies to a system. Therefore the migration to new hardware generations is complicated. Secondly, the execution time of single statements can vary depending on input data, nondeterministic scheduling, cache behavior, etc. Timing measurement can not guarantee that the actual timing will be within required boundaries. Finally, in applications like the design of asynchronous circuits, an arbitrary delay of signals can be expensive.

In real time verification, clock values usually are assumed to be nonnegative real, rational or natural numbers. As opposed to untimed systems, there is no generally accepted representation of sets or regions of timing values. Common tools use *difference bound matrices* [Dill 1989] and *clock regions* [Alur, Courcoubetis and Dill 1990, Alur 1991] to represent timing constraints. Real time systems often are modelled with timed automata [Alur 1998, Alur and Dill 1990] or timed transition systems [Henzinger, Manna and Pnueli 1992]. For an overview on real time logics and models, see [Alur and Henzinger 1992]. Reachability and model checking algorithms for these models are given in [Alur et al. 1990]. Generally, the complexity of verifying real-time systems is much higher than that for untimed systems. Moreover, timing constructs are often represented using an explicit state representation. Consequently, the number of states that can be handled is relatively small ($10^5 - 10^7$). Thus, at present, only highly abstracted examples (e.g., [Archer and Heitmeyer 1996]) can be verified automatically by model checking tools like KRONOS [Yovine 1997, Yovine 1998] or UPPAAL [Larsen, Petterson and Yi 1997, Aceto, Bergueno and Larsen 1998].

It is a challenging research task to find a paradigm separating the real-time component from the functional and reactive component in the specification of typical real-time requirements. This could make model checking an integral component in the development of reactive real-time systems.

15.3. Probabilistic Model Checking

Some safety-critical systems have a stochastic behavior. This may be either due to the fact that some part of the outside world, which is stochastic in nature, is modelled as part of the system, or because of hardware failures which may happen stochastically. Available model checkers usually model the probabilistic behavior of such systems non-deterministically, missing the ability to assess how probable some system behavior is.

A number of theoretical papers have been written on probabilistic verification. Efficient algorithms have been given by several authors; for example, there is an LTL model checking algorithm which is exponential in the size of the formula and polynomial in the size of the Markov chain [Courcoubetis and Yannakakis 1995]. However, currently there are no probabilistic model checking tools available which can verify systems of realistic size. The bottleneck is the construction of the state space and the necessity to solve huge systems of linear equations. A more efficient alternative could be to perform the probability calculations using Multi-Terminal Binary Decision Diagrams (MTBDDs).

MTBDDs [Bahar, Frohm, Gaona, Hachtel, Macii, Pardo and Somenzi 1993, Clarke, Fujita, McGeer, Yang and Zhao 1993] differ from BDDs in that the leaves may have values other than 0 and 1; in this case the leaves contain transition probabilities. MTBDDs can be used to represent D–valued matrices efficiently. Consider a $2^m \times 2^m$–matrix A. Its elements a_{ij}, can be viewed as the values of a function $f_A : \{0, \dots 2^m - 1\} \times \{0, \dots 2^m - 1\} \to D$, where $f_A(i,j) = a_{ij}$. Using the standard encoding $c : B^m \to \{0, \dots 2^m - 1\}$ of boolean sequences of length less than m into the integers, this function may be interpreted as a D–valued boolean function $f : B^m \to D$ where $f(x, y) = f_A(c(x), c(y))$ for $x = (x_0 \ldots x_{m-1})$ and $y = (y_0 \ldots y_{m-1})$. This transformation now allows matrices to be represented as MTBDDs. In order to obtain an efficient MTBDD–representation, the variables of f are permuted. Instead of the MTBDD for $f(x_0 \ldots x_{m-1}, y_0 \ldots y_{m-1})$, the MTBDD obtained from $f(x_0, y_0, x_0, y_0, \ldots x_{m-1}, y_{m-1})$ can be used. This convention imposes a recursive structure on the matrix from which efficient recursive algorithms for all standard matrix operations can be derived.

MTBDDs can be integrated with a symbolic model checker and have the potential to outperform other matrix representations because they are very compact. For example, in [Hachtel, Macii, Pardo and Somenzi 1996] symbolic algorithms were developed to perform steady-state probabilistic analysis for systems with finite state models of more than 10^{27} states. While it is difficult to provide precise time complexity estimates for probabilistic model checking using MTBDDs, the success of BDDs in practice indicates that this is likely to be a worthwhile approach.

The standard model used in probabilistic model checking are finite state discrete-time Markov chains ([Hansson and Jonsson 1989, Courcoubetis and Yannakakis 1995, Aziz, Singhal, Balarin, Brayton and Sangiovanni-Vincentelli 1995, Aziz, Sanwal, Singhal and Brayton 1996]). This model is a powerful notation for the dependability analysis of fault-tolerant real-time control systems, performance analysis of commercial computer systems and networks, and operation of automated manufacturing systems.

To specify properties of finite state discrete-time Markov chains, *Probabilistic Real Time Computation Tree Logic* (PCTL) was introduced in [Hansson and Jonsson 1989]. PCTL augments **CTL** with time and probability; it is a very expressive logic and offers simple model checking algorithms that can be implemented using symbolic techniques in a straightforward manner [Baier, Clarke, Hartonas-Garmhausen, Kwiatkowska and Ryan 1997].

However, in order to make model checking a standard method for probabilistic verification, more experiences with industrial size examples, typical requirements and efficient tools are necessary.

15.4. Model Checking for Security Protocols

Security protocols are another promising area for the application of model checking techniques. The increasing amount of confidential information (such as monetary transactions) sent over insecure communication links (such as the internet) requires more and more sophisticated encryption protocols. Like hardware designs, these protocols can have subtle bugs which are difficult to find. It may be possible to use the same exhaustive search techniques as in model checking to verify security protocols. By examining all possible execution traces of the protocol in the presence of a malicious adversary with well defined capabilities, it may be possible to determine if an attack on the protocol could be successful.

Typically, security protocols can be thought of as a set of principals which send messages to each other. The hope is that by requiring agents to produce a sequence of formatted and encrypted messages, the security goals of the protocol can be achieved. For example, if a principal A receives a message encrypted with a key known only by principal B, then principal A should be able to conclude that principal B created the message. However, it would be incorrect to conclude that principal A is talking to principal B. An adversary could be replaying a message overheard during a previous conversation between A and B. If the aim is to keep the message secret, then as long as the adversary does not learn the key, this security property is satisfied. If, however, the aim is to authenticate B to A, then clearly this is not satisfied since the message was not necessarily sent by B.

Since the reasoning behind the correctness of these protocols can be subtle, researchers have tried turning to formal methods to prove protocols correct. In [Burrows, Abadi and Needham 1989], a logic of belief is developed in which one could formally reason about security protocols by stating axioms about the protocol and trying to derive theorems about its security. [Kindred and Wing 1996] added some automation to this process by generating theory checkers for these logics. In [Meadows 1994], a different approach is taken by modelling a security protocol in terms of a set of rewrite rules. These rules capture the way that the adversary can learn new information using encryption and decryption, and by receiving replies to messages sent to participants of the protocol. In [Woo and Lam 1993], the authors propose a model for authentication and provide a number of inference rules that could be used for proving properties in this model. The paper [Mitchell, Mitchell and Stern 1997] investigated the use of Murφ, a previously existing model checker, for verifying security protocols.

A special purpose model checker for authentification protocols could contain two orthogonal components. The first is a *state exploration component*. Each honest agent can be described by the sequence of actions that it takes during a run of the protocol, and can be viewed as a finite-state machine. A trace of the actions performed by the asynchronous composition of these state machines corresponds to a possible execution of the protocol by the agents. By performing an exhaustive search of the state space of the composition, it can be determined if various security properties are violated.

The second component would be the *message derivation engine* which is used to model what the adversary is allowed to do. It can be implemented as a simple natural deduction theorem prover for constructing valid messages. The adversary can intercept messages, misdirect messages, and generate new messages using encryption, decryption, concatenation (pairing), and projection. Each time a message is sent, the adversary intercepts the message and adds it to the set of assumptions it can use to derive new messages. Whenever an honest agent receives a message, the message must have been generated by the derivation engine.

A first prototypical implementation shows that this framework can be successfully used to analyze threats and exhibit possible attacks in authentication protocols. It is also general enough to handle other kinds of security protocols such as key exchange and electronic commerce. Moreover, combining model checking with other automated deduction techniques could make it possible to verify both the encryption algorithm and the actual implementation at the same time. However, for a widespread use it is additionally necessary to integrate the model checking approach with other, more well-established security design methods.

Acknowledgments

We would like to thank Wolfgang Heinle for help with initial versions of this chapter, the editor for his patience with us during its preparation, and the referees for many useful comments and suggestions.

Bibliography

ABADI M., LAMPORT L. AND MERZ S. [1996], A TLA solution to the RPC–memory specification problem, *in* M. Broy, S. Merz and K. Spies, eds, 'Formal System Specification: The RPC–Memory Specification Case Study', Vol. 1169 of *LNCS*, Springer, pp. 21–66.

ACETO L., BERGUENO A. AND LARSEN K. [1998], Model checking via reachability testing for timed automata, *in* B. Steffen, ed., 'Proc. 4th Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98)', Vol. 1384 of *LNCS*, Springer, Lisbon, pp. 281–297.

AHO A. V., HOPCROFT J. E. AND ULLMAN J. D. [1974], *The Design and Analysis of Computer Algorithms*, Addison–Wesley.

AJTAI M. AND GUREVICH Y. [1987], 'Monotone versus positive', *Journal of the ACM* **34**, 1004–1015.

ALPERN B. AND SCHNEIDER F. [1985], 'Defining liveness', *Information Processing Letters* **21**, 181–185.

ALUR R. [1991], Techniques for Automatic Verification of Real–Time Systems, PhD thesis, Stanford University.

ALUR R. [1998], Timed automata, *in* 'Verification of Digital and Hybrid Systems', NATO ASI Summer School Series, Springer.

ALUR R., BRAYTON R. K., HENZINGER T., QUADEER S. AND RAJAMANI S. K. [1997], Partial-order reduction in symbolic state space exploration, *in* 'Proc. 9th Int. Conf. on Computer Aided Verification (CAV '97)', Vol. 1254 of *LNCS*, Springer, Haifa, Israel, pp. 340–351.

ALUR R., COURCOUBETIS C. AND DILL D. [1990], Model–checking for real–time systems, *in* 'Proc. 5th Ann. IEEE Symp. on Logic in Computer Science (LICS '90)', IEEE Comp. Soc. Press, pp. 414–425.

ALUR R. AND DILL D. [1990], Automata for modelling real–time systems, *in* 'Proc. 17th Int. Conf. on Automata, Languages and Programming (ICALP '90)', Vol. 443 of *LNCS*, Springer, pp. 322–335.

ALUR R. AND HENZINGER T. A. [1992], Logics and models of real–time: A survey, *in* 'Real–Time: Theory in Practice', LNCS, Springer.

ANDERSEN H. R. [1994], On model checking infinite–state systems, *in* A. Nerode and Matiyasevich, eds, 'Logic at St. Petersburg. Symp. on Logical Foundations of Computer Science (LFCS '94)', Vol. 813 of *LNCS*, Springer, St. Petersburg, Russia, July 11–14.

ANDERSEN H. R., STIRLING C. AND WINSKEL G. [1994], A compositional proof system for the modal μ–calculus, *in* 'Proc. 9th Ann. IEEE Symp. on Logic in Computer Science (LICS '94)', IEEE Computer Society Press, Paris, France, pp. 144–153. BRICS Report RS–94–34.

ANUCHITANUKUL A. [1995], Synthesis of Reactive Programs, PhD thesis, Stanford.

ARCHER M. AND HEITMEYER C. [1996], Mechanical verification of timed automata: A case study, *in* 'IEEE Real–Time Technology and Applications Symp. (RTAS'96)', IEEE Computer Society Press, Boston MA.

AZIZ A., SANWAL K., SINGHAL V. AND BRAYTON R. K. [1996], Verifying continuous Markov chains, *in* R. Alur and T. Henzinger, eds, 'Proc. 8th Workshop on Computer Aided Verification (CAV '96)', Vol. 1102 of *LNCS*, Springer, pp. 269–276.

AZIZ A., SINGHAL V., BALARIN F., BRAYTON R. K. AND SANGIOVANNI-VINCENTELLI A. L. [1995], It usually works – the temporal logic of stochastic systems, *in* P. Wolper, ed., 'Proc. 7th Workshop on Computer Aided Verification (CAV '95)', Vol. 939 of *LNCS*, Springer, pp. 155–166.

BAHAR R. I., FROHM E. A., GAONA C. M., HACHTEL G. D., MACII E., PARDO A. AND SOMENZI F. [1993], Algebraic decision diagrams and their applications, *in* 'Proc. Int. Conf. on Computer Aided Design (ICCAD '93)', Santa Clara, pp. 188–191.

Baier C., Clarke E. M., Hartonas-Garmhausen V., Kwiatkowska M. and Ryan M. [1997], Symbolic model checking for probabilistic processes, *in* 'Proc. Int. Conf. on Automata, Languages and Programming (ICALP '97))', Vol. 1256 of *LNCS*, pp. 430–437.

Ben-Ari M., Manna Z. and Pnueli A. [1983], 'The temporal logic of branching time', *Acta Informatica* **20**, 207–226.

Bensalem S., Bouajani A., Loiseaux C. and Sifakis J. [1992], Property preserving simulations, *in* G. V. Bochmann and D. K. Probst, eds, 'Proc. 4th Int. Conf. on Computer Aided Verification (CAV '92)'.

Berezin S., Clarke E. M., Jha S. and Marrero W. [1996], Model checking algorithms for the μ–calculus, Technical Report CMU–CS–96–180, CMU.

Bermann C. L. [1991], 'Circuit width, register allocation and ordered binary decision diagrams', *IEEE Trans. on Computer–Aided Design* **10**(8), 1059–1066.

Bern J., Meinel C. and Slobodovà A. [1995], Global rebuilding of BDDs – avoiding the memory requirement maxima, *in* P. Wolper, ed., 'Proc. 7th Workshop on Computer Aided Verification (CAV '95)', Vol. 939 of *LNCS*, Springer, pp. 4–15.

Bhat G. and Cleaveland R. [1996], Efficient local model checking for fragments of the modal μ–calculus, *in* T. Margaria and B. Steffen, eds, 'Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)', Vol. 1055 of *LNCS*, Springer, pp. 107–126.

Biere A. [1997], Effiziente Modellprüfung des μ–Kalküls mit binären Entscheidungsdiagrammen, PhD thesis, University of Karlsruhe, Germany.

Biere A., Cimatti A., Fujita M. and Zhu Y. [1999], Symbolic model checking using SAT procedures instead of BDDs, *in* 'Proc. 36th ACM/IEEE Design Automation Conference (DAC '99)'.

Biere A., Cimatti A. and Zhu Y. [1999], Symbolic model checking without BDDs, *in* 'Proc. Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)', Vol. 1579 of *LNCS*, Springer.

Bjørner N., Browne A., Chang E., Colón M., Kapur A., Manna Z., Simpa H. B. and Uribe T. E. [1995], STeP: The Stanford theorem prover – user's manual, Technical Report STAN–CS–TR–95–1562, Department of Computer Science, Stanford University.

Bjørner N., Browne A., Chang E., Colón M., Kapur A., Manna Z., Sipma H. B. and Uribe T. E. [1996], STeP: Deductive–algorithmic verification of reactive and real–time systems, *in* 'Proc. 8th Workshop Computer Aided Verification (CAV '96)', Vol. 1102 of *LNCS*, Springer.

Blackburn P., de Rijke M. and Venema Y. [2000], *Modal Logic*, Elsevier. draft, 395 pp.

Boigelot B. and Godefroid P. [1996], Symbolic verification of communication protocols with infinite state spaces using qdds, *in* R. Alur and T. Henzinger, eds, 'Proc. 8th Workshop on Computer Aided Verification (CAV '96)', Vol. 1102 of *LNCS*, Springer, pp. 1–12.

Borälv A. [1997], The industrial success of verification tools based on stålmarck's method, *in* O. Grumberg, ed., 'Proc. 9th Workshop on Computer Aided Verification (CAV '97)', Vol. 1254 of *LNCS*, Springer.

Brace K. S., Rudell R. L. and Bryant R. E. [1990], Efficient implementation of a BDD package, *in* 'Proc. 27th ACM/IEEE Design Automation Conference (DAC '90)', pp. 40–45.

Bradfield J. and Stirling C. [1991], Local model checking for infinite state spaces, *in* 'Proc. 3rd Workshop on Computer Aided Verification (CAV '91)', LNCS, Springer.

Browne M. C. and Clarke E. M. [1986], SML: A high level language for the design and verification of finite state machines, *in* 'IFIP WG 10. 2 Int. Working Conf. from HDL Descriptions to Guaranteed Correct Circuit Designs', IFIP, Grenoble, France.

Browne M. C., Clarke E. M. and Dill D. [1985], Checking the correctness of sequential circuits, *in* 'Proc. 1985 Int. IEEE Conf. on Computer Design', IEEE, Port Chester, New York.

Browne M. C., Clarke E. M. and Dill D. [1986], Automatic circuit verification using temporal logic: Two new examples, *in* 'Formal Aspects of VLSI Design', Elsevier Science Publishers (North Holland).

BROWNE M. C., CLARKE E. M. AND GRUMBERG O. [1988], 'Characterizing finite Kripke structures in propositional temporal logic', *Theoretical Computer Science* **59**(1–2), 115–131.

BROWNE M. C., CLARKE E. M. AND GRUMBERG O. [1989], 'Reasoning about networks with many identical finite–state processes', *Information and Computation* **81**(1), 13–31.

BROWNE M., CLARKE E. M., DILL D. AND MISHRA B. [1986], 'Automatic verification of sequential circuits using temporal logic', *IEEE Trans. on Computers* **C-35**(12), 1035–1044.

BRYANT R. E. [1986], 'Graph–based algorithms for Boolean function manipulation', *IEEE Trans. on Computers* **C–35**(8), 677–691.

BRYANT R. E. [1991], 'On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication', *IEEE Trans. on Computers* **40**(2), 205–213.

BRYANT R. E. [1992], 'Symbolic Boolean manipulation with ordered binary decision diagrams', *ACM Computing Surveys* **24**(3), 293–317.

BÜCHI J. R. [1962], On a decision method in restricted second order arithmetic, *in* 'Proc. Int. Congr. Logic, Method and Philosophy of Science 1960', Stanford University Press, Palo Alto, CA, USA, pp. 1–12.

BURCH J. R., CLARKE E. M., DILL D., LONG D. E. AND MCMILLAN K. L. [1994], 'Symbolic model checking for sequential circuit verification', *IEEE Trans. on Computer Aided Design of Integrated Circuits* **13**(4), 401–424.

BURCH J. R., CLARKE E. M., GRUMBERG O., LONG D. E. AND MCMILLAN K. L. [1992], 'Automatic verification of sequential circuit designs', *Phil. Trans. R. Soc. Lond. A* **339**, 105–120.

BURCH J. R., CLARKE E. M. AND LONG D. E. [1991*a*], Representing circuits more efficiently in symbolic model checking, *in* 'Proc. 28th ACM/IEEE Design Automation Conference (DAC '91)'.

BURCH J. R., CLARKE E. M. AND LONG D. E. [1991*b*], Symbolic model checking with partitioned transition relations, *in* A. Halaas and P. B. Denyer, eds, 'Proc. Int. Conf. on Very Large Scale Integration (VLSI '91)', Edinburgh, Scotland.

BURCH J. R., CLARKE E. M., MCMILLAN K. L., DILL D. AND HWANG L. J. [1992], 'Symbolic model checking: 10^{20} states and beyond', *Information and Computation* **98**(2), 142–170. also in 5th IEEE LICS 90.

BURCH J. R., CLARKE E. M., MCMILLAN K. L. AND DILL D. L. [1990], Sequential circuit verification using symbolic model checking, *in* 'Proc. 27th ACM/IEEE Design Automation Conference (DAC '90)'.

BURGESS J. [1984], Basic tense logic, *in* F. G. D. Gabbay, ed., 'Handbook of Philosophical Logic', Reidel, chapter II. 2, pp. 89–134.

BURKART O. AND ESPARZA J. [1997], 'More infinite results', *Electronic Notes in Theoretical Computer Science* **6**. http://www. elsevier. nl/locate/entcs/volume6. html.

BURROWS M., ABADI M. AND NEEDHAM R. [1989], A logic of authentication, Technical Report 39, DEC Systems Research Center.

BURSTALL M. [1974], Program proving as hand simulation with a little induction, *in* 'Proc. IFIP Congress, Stockholm', North Holland, pp. 308–312.

CHANDRA A. AND HAREL D. [1980], 'Computable queries for relational databases', *J. of Computer and System Sciences* **21**, 156–178.

CLARKE E. M. AND DRAGHICESCU I. A. [1988], Expressibility results for linear time and branching time logics, *in* 'Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency', Vol. 354 of *LNCS*, Springer, pp. 428–437.

CLARKE E. M., DRAGHICESCU I. A. AND KURSHAN R. P. [1990], A unified approach for showing language containment and equivalence between various types of ω–automata, *in* A. Arnold and N. D. Jones, eds, 'Proc. 15th Coll. on Trees in Algebra and Programming', Vol. 407 of *LNCS*, Springer.

Clarke E. M. and Emerson E. A. [1981], Synthesis of synchronization skeletons for branching time temporal logic, *in* 'Proc. Workshop on Logic of Programs', Vol. 131 of *LNCS*, Springer, Yorktown Heights, NY.

Clarke E. M., Emerson E. A. and Sistla A. P. [1986], 'Automatic verification of finite–state concurrent systems using temporal logic specifications', *ACM Transactions on Programming Languages and Systems* **8**(2), 244–263.

Clarke E. M., Filkorn T. and Jha S. [1993], Exploiting symmetry in temporal logic model checking, *in* C. Courcoubetis, ed., 'Proc. 5th Workshop on Computer Aided Verification (CAV '93)', Vol. 697 of *LNCS*, Springer, Elounda, Crete.

Clarke E. M., Fujita M. and Heinle W. [1997], Hybrid spectral transform diagrams, *in* 'Proc. 1st Int. Conf. on Information, Communications and Signal Processing (ICICS '97)'.

Clarke E. M., Fujita M., McGeer P., Yang J. and Zhao X. [1993], Multi–terminal binary decision diagrams: An efficient data structure for matrix representation, *in* 'Proc. Int. Workshop on Logic Synthesis (IWLS '93)', Tahoe City.

Clarke E. M., Fujita M. and Zhao X. [1995], Hybrid decision diagrams —- overcoming the limitations of MTBDDs and BMDs, *in* 'Proc. IEEE Int. Conf. on Computer Aided Design (ICCAD '95)', IEEE Computer Society Press, pp. 54–60.

Clarke E. M., Fujita M. and Zhao X. [1996], Multi–terminal binary decision diagrams and hybrid decision diagrams, *in* T. Sasao and M. Fujita, eds, 'Representations of Discrete Functions', Kluwer academic publishers, chapter 4, pp. 93–108.

Clarke E. M. and Grumberg O. [1987*a*], Avoiding the state explosion problem in temporal model checking algorithms, *in* 'Proc. 6th Ann. ACM Symp. on Principles of Distributed Computing', pp. 294–303.

Clarke E. M. and Grumberg O. [1987*b*], Research on automatic verification of finite–state concurrent systems, Technical Report CMU-CS-87-105, Carnegie Mellon University.

Clarke E. M., Grumberg O. and Browne M. C. [1986], Reasoning about networks with many identical finite-state processes, *in* 'Proc. 5th Ann. ACM Symp. on Principles of Distributed Computing', ACM, pp. 240–248.

Clarke E. M., Grumberg O. and Hamaguchi K. [1997], 'Another look at LTL model checking', *Formal Methods in System Design* **10**, 47–71.

Clarke E. M., Grumberg O., Hiraishi H., Jha S., Long D. E., McMillan K. L. and Ness L. A. [1993], Verification of the Futurebus+ cache coherence protocol, *in* L. Claesen, ed., 'Proc. 11th Int. Symp. on Computer Hardware Description Languages and their Applications', North–Holland.

Clarke E. M., Grumberg O. and Jha S. [1995], Parametrized networks, *in* S. Smolka and I. Lee, eds, 'Proc. 6th Int. Conf. on Concurrency Theory (CONCUR '95)', Vol. 962 of *LNCS*, Springer.

Clarke E. M., Grumberg O. and Long D. E. [1993], Model checking, *in* M. Broy, ed., 'Deductive Program Design', Springer NATO ASI series F, pp. 305–350.

Clarke E. M., Grumberg O. and Long D. E. [1994*a*], 'Model checking and abstraction', *ACM Transactions on Programming Languages and Systems* **16**(5), 1512–1542. also in 19th ACM POPL '92.

Clarke E. M., Grumberg O. and Long D. E. [1994*b*], Verification tools for finite–state concurrent systems, *in* J. W. de Bakker, W. P. de Roever and G. Rozenberg, eds, 'A Decade of Concurrency – Reflections and Perspectives', Vol. 803 of *LNCS*, Springer, pp. 124–175. REX School/Symposium, Nordwijkerhout, The Netherlands, June 1993.

Clarke E. M., Grumberg O., McMillan K. and Zhao X. [1994], Efficient generation of counterexamples and witnesses in symbolic model checking, Technical Report CMU–CS–94–204, Carnegie Mellon University, Pittsburgh.

Clarke E. M., Grumberg O., Minea M. and Peled D. [1999], 'State space reductions using partial order techniques', *Int. Journal on Software Tools for Technology Transfer* . to appear.

Clarke E. M., Grumberg O. and Peled D. [1999], *Model Checking*, MIT Press, Boston, MA.

CLARKE E. M., JHA S., LU Y. AND MINEA M. [1997], Equivalence checking using abstract BDDs. manuscript.

CLARKE E. M., KHAIRA K. AND ZHAO X. [1993], Word level model checking — a new approach for verifying arithmetic circuits, *in* 'Proc. 30th ACM/IEEE Design Automation Conference (DAC '93)', IEEE Computer society press.

CLARKE E. M., KIMURA S., LONG D. E., MICHAYLOV S., SCHWAB S. A. AND VIDAL J. P. [1992], Symbolic computation algorithms on shared memory multiprocessors, *in* N. Suzuki, ed., 'Shared Memory Multiprocessing', MIT Press, pp. 53–80.

CLARKE E. M., LONG D. E. AND MCMILLAN K. L. [1989], Compositional model checking, *in* 'Proc. 4th Ann. IEEE Symp. on Logic in Computer Science (LICS '89)', Asilomar, Calif.

CLARKE E. M., LONG D. E. AND MCMILLAN K. L. [1991], 'A language for compositional specification and verification of finite state hardware controllers', *Proc. IEEE* **79**(9), 1283–1292.

CLARKE E. M., MCMILLAN K. L., ZHAO X., FUJITA M. AND YANG J. [1993], Spectral transforms for large Boolean functions with applications to technology mapping, *in* 'Proc. 30th ACM/IEEE Design Automation Conference (DAC '93)', IEEE Computer Society Press, pp. 54–60.

CLARKE E. M. AND MISHRA B. [1984], Automatic verification of asynchronous circuits, *in* 'Proc. Workshop on Logics of Programs', Vol. 164 of *LNCS*, Springer, pp. 101–115.

CLARKE E. M. AND ZHAO X. [1994], Combining symbolic computation and theorem proving: some problems of Ramanujan, *in* A. Bundy, ed., '12th Int. Conf. on Automated Deduction (CADE '94)', Vol. 814 of *LNCS*, Springer, Nancy, France, pp. 758–763.

CLEAVELAND R. [1990], 'Tableau–based model checking in the propositional μ–calculus', *Acta Informatica* **27**(8), 725–747.

CLEAVELAND R. AND STEFFEN B. [1993], 'A linear–time model–checking algorithm for the alternation–free modal μ–calculus', *Formal Methods in System Design* **2**(2), 121–147.

COURCOUBETIS C., VARDI M. Y., WOLPER P. AND YANNAKAKIS M. [1992], 'Memory efficient algorithms for the verification of temporal properties', *Formal Methods in System Design* **1**, 275–288.

COURCOUBETIS C. AND YANNAKAKIS M. [1995], 'The complexity of probabilistic verification', *Journal of the ACM* **42**(4), 857–907.

COUSOT P. AND COUSOT R. [1977], Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, *in* 'Proc. 4th Ann. ACM Symp. on Principles of Programming Languages (POPL '77)'.

COUSOT P. AND COUSOT R. [1979], Systematic design of program analysis frameworks, *in* 'Proc. 6th Ann. ACM Symp. on Principles of Programming Languages (POPL '79)'.

DAM M. [1994], 'CTL* and ECTL* as fragments of the modal μ–calculus', *Theoretical Computer Science* **126**, 77–96.

DAM M. [1995], Compositional proof systems for model checking infinite state processes, *in* 'Proc. 6th Int. Conf. on Concurrency Theory (CONCUR '95)', Vol. 962 of *LNCS*, Springer, pp. 12–26.

DAMS D. [1995], Abstract interpretation and partition refinement for model checking, PhD thesis, Technical University Eindhoven.

DAMS D., GRUMBERG O. AND GERTH R. [1994], Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*, *in* E.-R. Olderog, ed., 'Programming Concepts, Methods and Calculi (PROCOMET '94)', IFIP Transactions, North Holland / Elsevier, Amsterdam, pp. 561 – 581.

DAVEY A. A. AND PRIESTLEY H. A. [1990], *Introduction to Lattices and Order*, Cambridge Mathematical Textbooks, Cambridge University Press.

DAWAR A., LINDELL S. AND WEINSTEIN S. [1996], First order logic, fixed point logic and linear order, *in* H. Kleine-Büning, ed., 'Proc. Computer Science Logic (CSL '95)', Vol. 1092 of *LNCS*, Springer, pp. 161–177.

DEROEVER, W., LANGMAACK, H. AND PNUELI, A., EDS [1998], *Compositionality: The Significant Difference*, Vol. 1536 of *LNCS*, Springer.

Dill D. [1989], Timing assumptions and verification of finite–state concurrent systems, *in* J. Sifakis, ed., 'Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems', Vol. 407 of *LNCS*, Springer, Grenoble, France, pp. 197–212.

Dill D. L. and Clarke E. M. [1986], 'Automatic verification of asynchronous circuits using temporal logic', *IEEE Proceedings* **133**(5).

Dingel J. and Filkorn T. [1995], Model checking for infinite state systems using data abstraction, assumption–commitment style reasoning and theorem proving, *in* P. Wolper, ed., 'Proc. 7th Workshop on Computer Aided Verification (CAV '95)', Vol. 939 of *LNCS*, Springer, pp. 45–69.

Edelkamp S. and Reffel F. [1998], OBDDs in heuristic search, *in* O. Herzog and A. Günter, eds, 'Proc. KI-98: Advances in Artificial Intelligence', Vol. 1504 of *LNCS/LNAI*, Springer, pp. 81–92.

Ehrenfeucht A. [1961], 'An application of games to the completeness problem for formalized theories', *Fund. Math.* **49**, 129–141.

Emerson E. A. [1985], Automata, tableaux, and temporal logic, *in* R. Parikh, ed., 'Proc. Int. Conf. on Logics of Programs', Vol. 193 of *LNCS*, Springer, pp. 79–88.

Emerson E. A. [1990], Temporal and modal logic, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B, Elsevier, pp. 997–1072.

Emerson E. A. and Clarke E. M. [1980], Characterizing correctness properties of parallel programs using fixpoints, *in* 'Proc. 17th Int. Coll. on Automata, Languages and Programming (ICALP '80)', Vol. 85 of *LNCS*, EATCS, Springer, pp. 169–181.

Emerson E. A. and Clarke E. M. [1982], 'Using branching time logic to synthesize synchronization skeletons', *Science of Computer Programming* **2**, 241–266.

Emerson E. A. and Halpern J. Y. [1985], 'Decision procedures and expressiveness in the temporal logic of branching time', *Journal of Computer and System Sciences* **30**(1), 1–24.

Emerson E. A. and Halpern J. Y. [1986], '"sometimes" and "not never" revisited: on branching time vs. linear time', *Journal of the ACM* **33**, 151–178.

Emerson E. A., Jutla C. S. and Sistla A. P. [1993], On model–checking for fragments of μ–calculus, *in* C. Courcoubetis, ed., 'Proc. 5th Workshop on Computer Aided Verification (CAV '93)', Vol. 697 of *LNCS*, Springer.

Emerson E. A. and Lei C. L. [1985], Modalities for model checking: Branching time strikes back, *in* 'Proc. 12th Symp. on Principles of Programming Languages (POPL '85)', New Orleans, La.

Emerson E. A. and Lei C. L. [1986], Efficient model checking in fragments of the propositional μ–calculus, *in* 'Proc. 1st Symp. on Logic in Computer Science (LICS '86)', Boston, Mass.

Emerson E. A. and Sistla A. P. [1984], 'Deciding full branching time logic', *Information and Control* **61**, 175–201.

Emerson E. A. and Sistla A. P. [1993], Symmetry and model checking, *in* C. Courcoubetis, ed., 'Proc. 5th Workshop on Computer Aided Verification (CAV '93)', Vol. 697 of *LNCS*, Springer, Elounda, Crete.

Enders R., Filkorn T. and Taubner D. [1993], 'Generating BDDs for symbolic model checking', *Distributed Computing* **6**, 155–164.

Esparza J. [1994], 'Model checking using net unfoldings', *Science of Computer Programming* **23**(2–3), 151–195.

Felt E., York G., Brayton R. and Vincentelli A. S. [1993], Dynamic variable reordering for BDD minimization, *in* 'Proc. European Design Automation Conference (EuroDAC '93)', pp. 130–135.

Fischer M. J. and Ladner R. E. [1979], 'Propositional dynamic logic of regular programs', *Journal of Computer and System Sciences* **18**(2), 194–211.

Fitting M. [1983], *Proof methods for modal and intuitionistic logics*, Reidel, Dordrecht.

Fraïssé R. [1954], Sur quelques classifications des systèmes de relations, Séries A 1, Publications Scientifiques de l'Universit'e d'Algerie.

FRANCEZ N. [1986], *Fairness*, Text and Monographs in Computer Science, Springer.

FUJI H., OOTOMO G. AND HORI C. [1993], Interleaving based variable ordering methods for ordered binary decision diagrams, *in* 'Proc. Int. Conf. on Computer Aided Design (ICCAD '93)', IEEE.

GABBAY D. [1989], The declarative past and imperative future: Executable temporal logic for interactive systems, *in* B. Banieqbal, ed., 'Temporal Logic in Specification', Vol. 398 of *LNCS*, Springer, pp. 431–448.

GABBAY D., HODKINSON I. AND REYNOLDS M. [1994], *Temporal Logic: Mathematical Foundations and Computational Aspects*, Vol. 1, Clarendon Press, Oxford.

GABBAY D., PNUELI A., SHELAH S. AND STAVI J. [1980], On the temporal analysis of fairness, *in* 'Proc. 7th ACM Symp. on Principles of Programming Languages (POPL '80)', pp. 163–173.

GERMAN S. M. AND SISTLA A. P. [1992], 'Reasoning about systems with many processes', *Journal of the ACM* **39**, 675–735.

GERTH R., KUIPER R., PELED D. AND PENCZEK W. [1995], A partial order approach to branching time logic model checking, *in* 'Proc. 3rd Israel Symp. on the Theory of Computing and Systems (ISTCS '95)', IEEE Computer Society Press, pp. 130–140.

GODEFROID P. [1990], Using partial orders to improve automatic verification methods, *in* 'Proc. 2nd Workshop on Computer Aided Verification (CAV '90)', Vol. 531 of *LNCS*, Springer, Rutgers, New Brunswick, pp. 176–185.

GODEFROID P., HOLZMANN G. J. AND PIROTTIN D. [1995], 'State–space caching revisited', *Formal Methods in System Design* **7**(3), 1–15.

GODEFROID P. AND LONG D. E. [1996], Symbolic protocol verification with queue BDDs, *in* 'Proc. 11th Ann. IEEE Symp. on Logic in Computer Science (LICS '96)', New Brunswick.

GODEFROID P. AND PIROTTIN D. [1993], Refining dependencies improves partial-order verification methods, *in* 'Proc. 5th Workshop on Computer Aided Verification (CAV '93)', Vol. 697 of *LNCS*, Springer, Elounda, Crete, pp. 438–449.

GODEFROID P. AND WOLPER P. [1991], A partial approach to model checking, *in* 'Proc. 6th Ann. IEEE Symp. on Logic in Computer Science (LICS '91)', Amsterdam, pp. 406–415.

GRAF S. AND STEFFEN B. [1990], Compositional minimization of finite state processes, *in* R. Kurshan and E. M. Clarke, eds, 'Proc. 2nd Workshop on Computer Aided Verification (CAV '90)', Vol. 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society. Also in Springer LNCS 531.

GRUMBERG O. AND LONG D. E. [1994], 'Model checking and modular verification', *ACM Transactions on Programming Languages and Systems* **16**, 843–872.

GUNTER C. A. AND SCOTT D. S. [1990], Semantic domains, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B, Elsevier, pp. 633–674.

GUREVICH Y. AND SHELAH S. [1986], 'Fixed–point extensions of first–order logic', *Annals of Pure and Applied Logic* **32**, 265–280.

GUROV D., BEREZIN S. AND KAPRON B. M. [1996], A modal μ–calculus and a proof system for value passing processes, *in* 'Proc. Int. Workshop on Verification of Infinite State Systems (INFINITY '96)', Electronic Notes in Theoretical Computer Science, Pisa, pp. 149–163. Report, University of Passau, MIP–9614 July 1996.

HACHTEL G. D., MACII E., PARDO A. AND SOMENZI F. [1996], 'Markovian analysis of large finite state machines', *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems* **15**(12), 1479–1493.

HANSSON H. AND JONSSON B. [1989], A framework for reasoning about time and reliability, *in* 'Proc. 10th IEEE Real-Time Systems Symp.', pp. 102–111.

HAREL D. [1984], Dynamic logic, *in* F. G. D. Gabbay, ed., 'Handbook of Philosophical Logic', Reidel, chapter II. 10, pp. 488–604.

HAREL D. AND PNUELI A. [1985], On the development of reactive systems, *in* K. R. Apt, ed., 'Logics and Models of Concurrent Systems', Vol. F13 of *NATO ASI Series*, Springer, pp. 477–498.

HAR'EL Z. AND KURSHAN R. P. [1990], 'Software for analytical development of communications protocols', *AT&T Tech. J.* **69**(1), 45–59.

HENZINGER T., MANNA Z. AND PNUELI A. [1992], Timed transition systems, Technical Report TR 92–1263, Dept. of CS, Cornell Univ.

HOLZMANN G. [1991], *Design and Validation of Computer Protocols*, Software Series, Prentice Hall.

HOLZMANN G. [1995], An analysis of bitstate hashing, *in* 'Proc. 15th Int. Conf. on Protocol Specification, Testing and Verification', INWG/IFIP, Chapman and Hall, Warsaw, pp. 301–314.

HOPCROFT J. E. AND ULLMAN J. D. [1979], *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

HUGHES G. E. AND CRESSWELL M. J. [1977], *An Introduction to Modal Logic*, Methuen.

IEEE [1994], *IEEE Standard for the Futurebus+ Logical Protocol Specification*, IEEE Computer Society. IEEE Standard 896. 1, 1994 Edition.

IMMERMAN N. [1986], 'Relational queries computable in polynomial time', *Information and Control* **68**, 86–104.

IP C. W. AND DILL D. [1993], Better verification through symmetry, *in* L. Claesen, ed., 'Proc. 11th Int. Symp. on Computer Hardware Description Languages and their Applications', North–Holland.

JANIN D. AND WALUKIEWICZ I. [1996], On the expressive completeness of the propositional μ–calculus with respect to monadic second order logic, *in* 'Proc. 7th Int. Conf. on Concurrency Theory (CONCUR '96)', Vol. 1119 of *LNCS*, Springer, Pisa, Italy.

JOSKO B. [1993], 'Modular specification and verification of reactive systems', Habilitationsschrift, University of Oldenburg.

KAMP H. W. [1968], Tense Logic and the Theory of Linear Order, PhD thesis, Univ. of Calif. , Los Angeles.

KANNAN R. AND LIPTON R. J. [1986], 'Polynomial–time algorithm for the orbit problem', *Journal of the ACM* **33**(4), 808–821.

KATZ S. AND PELED D. [1988], An efficient verification method for parallel and distributed programs, *in* deBakker et al., ed., 'Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency', Vol. 354 of *LNCS*, Springer.

KESTEN Y. AND PNUELI A. [1995], A complete axiomatization of qptl, *in* 'Proc. 10th Ann. IEEE Symp. on Logic in Computer Science (LICS '95)', pp. 2–12.

KIMURA S. [1995], Residue BDD and its application to the verification of arithmetic circuits, *in* 'Proc. 32nd Int. Design Automation Conference (DAC '95)'.

KINDRED D. AND WING J. M. [1996], Fast, automatic checking of security protocols, *in* 'USENIX 2nd Workshop on Electronic Commerce'.

KOZEN D. [1983], 'Results on the propositional μ–calculus', *Theoretical Computer Science* **27**, 333–354.

KOZEN D. AND PARIKH R. [1983], A decision procedure for the propositional μ–calculus, *in* 'Proc. Int. Symp. Logic of Programs'.

KOZEN D. AND TIURYN J. [1990], Logics of programs, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B, Elsevier, pp. 791–839.

KRIPKE S. A. [1963], 'Semantical considerations on modal logic', *Acta Philosophica Fennica* **16**, 83–94.

KRIPKE S. A. [1975], 'Outline of a theory of truth', *Journal of Philosophy* **72**, 690–716.

KRÖGER F. [1978], A uniform logical basis for the description, specification and verification of programs, *in* 'Proc. IFIP Work. Conf. Formal Desription of Programming Concepts', North Holland, St. Andrews, Canada, pp. 441–457.

KRÖGER F. [1987], *Temporal logic of Programs*, number 8 *in* 'EATCS monographs on TCS', Springer.

KUPFERMAN O. AND VARDI M. Y. [1996], Verification of fair transition systems, *in* R. Alur and T. Henzinger, eds, 'Proc. 8th Workshop on Computer Aided Verification (CAV '96)', Vol. 1102 of *LNCS*, Springer, pp. 372–382.

KURSHAN R. P. [1989], Analysis of discrete event coordination, *in* J. W. de Bakker, W. P. de Roever and G. Rozenberg, eds, 'Proc. REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness', Vol. 430 of *LNCS*, Springer.

KURSHAN R. P. [1994], *Computer–Aided Verification of Coordinating Processes: The Automata–Theoretic Approach*, Princeton University Press, Princeton, New Jersey.

KURSHAN R. P., LEVIN V., MINEA M., PELED D. AND YENIGUN H. [1997], Verifying hardware in its software context, *in* 'Proc. Int. Conf. on Computer Aided Design (ICCAD '97)', IEEE, San Jose, CA, USA.

KURSHAN R. P. AND MCMILLAN K. L. [1989], A structural induction theorem for processes, *in* 'Proc. 8th Ann. ACM Symp. on Principles of Distributed Computing', ACM Press.

LAMPORT L. [1980], "sometimes" is sometimes "not never", *in* 'Proc. 7th Ann. ACM Symp. on Principles of Programming Languages (POPL '80)', ACM, Las Vegas, pp. 174–185.

LAMPORT L. [1983], What good is temporal logic?, *in* 'Proc. IFIP', pp. 657–668.

LARSEN K., PETTERSON P. AND YI W. [1997], 'Uppaal in a nutshell', *Software Tools for Technology Transfer* **1**(1/2).

LEHMANN D., PNUELI A. AND STAVI J. [1981], Impartiality, justice, and fairness: The ethics of concurrent termination, *in* 'Proc. Int. Conf. on Automata, Languages, and Programming (ICALP '81)', Vol. 115 of *LNCS*, Springer.

LEWIS C. I. [1912], 'Implication and the algebra of logic', *Mind* **21**, 522–531.

LICHTENSTEIN O. AND PNUELI A. [1985], Checking that finite state concurrent programs satisfy their linear specification, *in* 'Proc. 12th Ann. ACM Symp. on Principles of Programming Languages (POPL '85)', ACM press, New Orleans, La.

LICHTENSTEIN O., PNUELI A. AND ZUCK L. [1985], The glory of the past, *in* 'Proc. Int. Conf. Logics of Programs', Vol. 193 of *LNCS*, Springer, pp. 196–218.

LOISEAUX C., GRAF S., SIFAKIS J., BOUAJJANI A. AND BENSALEM S. [1995], 'Property preserving abstractions for the verification of concurrent systems', *Formal Methods in System Design* **6**(1), 11–44. also in CAV '92, LNCS 663.

LONG D. E. [1993], Model Checking, Abstraction and Compositional Verification, PhD thesis, CMU School of Computer Science, CMU–CS–93–178.

LONG D. E., BROWNE A., CLARKE E. M., JHA S. AND MARRERO W. R. [1994], An improved algorithm for the evaluation of fixpoint expressions, *in* 'Proc. 6th Workshop on Computer Aided Verification (CAV '94)', LNCS, Springer, pp. 338–350.

MADER A. [1992], Tableau recycling, *in* 'Proc. 4th Workshop on Computer Aided Verification (CAV '92)', LNCS, Springer.

MANNA Z. AND PNUELI A. [1981], 'Verification of concurrent programs: The temporal framework'.

MANNA Z. AND PNUELI A. [1982*a*], Verification of concurrent programs: Temporal proof principles, *in* D. Kozen, ed., 'Proc. Workshop on Logics of Programs', Vol. 131 of *LNCS*, Springer, pp. 200–252.

MANNA Z. AND PNUELI A. [1982*b*], Verification of concurrent programs: The temporal framework, *in* R. S. Boyer and J. S. Moore, eds, 'The Correctness Problem in Computer Science', Academic Press, London, pp. 215–273.

MANNA Z. AND PNUELI A. [1987], A hierarchy of temporal properties, *in* 'Proc. 6th Ann. ACM Symp. on Principles of Distributed Computing', Stanford University Press, Stanford, CA 94305.

MANNA Z. AND PNUELI A. [1989], The anchored version of the temporal framework, *in* 'Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency', Vol. 354 of *LNCS*, Springer, pp. 201–284.

MANNA Z. AND PNUELI A. [1992], *The Temporal Logic of Reactive and Concurrent Systems – Specification*, Springer.

MANNA Z. AND PNUELI A. [1995], *Temporal Verifications of Reactive Systems – Safety*, Springer.

MARELLY R. AND GRUMBERG O. [1991], Gormel — grammar oriented model checker, Technical Report 697, The Technion.

MCMILLAN K. [1992], Using unfoldings to avoid the state space explosion problem in the verification of asynchronous circuits, *in* 'Proc. 4th Workshop on Computer Aided Verification', Vol. 663 of *LNCS*, Springer, Montreal, Canada, pp. 164–177.

MCMILLAN K. L. [1993], *Symbolic Model Checking*, Kluwer Academic Publishers.

MCMILLAN K. L. AND SCHWALBE J. [1992], Formal verification of the Encore Gigamax cache consistency protocol, *in* N. Suzuki, ed., 'Proc. Int. Symp. on Shared Memory Multiprocessing', MIT Press, pp. 111–134.

MEADOWS C. [1994], The NRL protocol analyzer: An overview, *in* 'Proc. 2nd Int. Conf. on the Practical Applications of Prolog'.

MERZ S. [1997], Abstraction as a proof rule, Technical report, Universität München.

MILNER R. [1980], *A Calculus of Communicating Systems*, Vol. 92 of *LNCS*, Springer.

MINEA M. [1999], Partial order reduction for model checking of timed automata, *in* 'Proc. Concur 99', LNCS, Springer.

MISHRA B. AND CLARKE E. M. [1985], 'Hierarchical verification of asynchonous circuits using temporal logic', *Theoretical computer science* **38**, 269–291.

MITCHELL J. C., MITCHELL M. AND STERN U. [1997], Automated analysis of cryptographic protocols using Murφ, *in* 'Proc. 1997 IEEE Symp. on Security and Privacy', IEEE Computer Society Press.

MOORE J. S. [1994], 'Introduction to OBDD algorithm for the ATP community', *Journal of automated reasoning* **12**, 33–45.

NIWINSKY D. [1988], Fixed points vs. infinite generation, *in* 'Proc. 3rd Ann. IEEE Symp. on Logic in Computer Science (LICS '88)', pp. 402–409.

NOWICKI J. R. AND ADAM L. J. [1990], *Digital Circuits*, Edward Arnold.

PAIGE R. AND TARJAN R. [1987], 'Three efficient algorithms based on partition refinement', *SIAM journal on computing* **16**(6).

PARK D. M. [1974], Finiteness is mu–ineffable, Theory of Computation Report 3, University of Warwick.

PARK D. M. [1981], Concurrency and automata on infinite sequences, *in* P. Deussen, ed., 'Theoretical Computer Science: 5th GI–Conference, Karlsruhe', Vol. 104 of *LNCS*, Springer, pp. 167–183.

PELED D. [1993], All from one, one for all: on model checking using representatives, *in* C. Courcoubetis, ed., 'Proc. 5th Workshop Computer Aided Verification (CAV '93)', Vol. 697, Springer, Elounda, Crete.

PELED D. AND WILKE T. [1997], 'Stutter–invariant temporal properties are expressible without the nexttime operator', *Information Processing Letters* .

PNUELI A. [1977], The temporal logic of programs, *in* 'Proc. 18th Ann. IEEE Symp. on Found. of Comp. Science (FOCS '77)', pp. 46–57.

PNUELI A. [1981], 'The temporal semantics of concurrent programs', *Theoretical Computer Science* **13**, 45–60.

PNUELI A. [1984], In transition from global to modular temporal reasoning about programs, *in* K. R. Apt, ed., 'Logics and Models of Concurrent Systems', Vol. 13 of *NATO ASI series*, Springer.

PRATT V. [1976], Semantical considerations on Floyd-Hoare logic, *in* 'Proc. 17th IEEE Symp. on Foundations of Comp. Sci. (FOCS'76)', pp. 109–121.

PRATT V. [1981], A decidable μ–calculus, *in* 'Proc. Ann. ACM Symp. on Foundations of Computer Science (FOCS '81)'.

PRIOR A. [1967], *Past, Present and Future*, Clarendon Press, Oxford.

PRIOR A. N. [1957], *Time and Modality*, Oxford University Press.

QUIELLE J. P. AND SIFAKIS J. [1981], Specification and verification of concurrent systems in CESAR, *in* 'Proc. 5th Int. Symp. on Programming'.

QUIELLE J. P. AND SIFAKIS J. [1982], Fairness and related properties in transition systems, Technical Report 292, IMAG.

RABIN M. O. [1969], 'Decidability of second order theories and automata on infinite trees', *Trans. AMS* **141**, 1–35.

REISIG W. [1998], *Elements of Distributed Algorithms*, Springer.

RESCHER N. AND URQUHART A. [1971], *Temporal Logic*, Springer.

RUDELL R. L. [1993], Dynamic variable reordering for ordered binary decision diagrams, *in* 'Proc. IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD '93)', pp. 42–47.

SAFRA S. [1988], On the complexity of omega–automata, *in* 'Proc. 29th IEEE Symp. on Foundations of Computer Science (FOCS '88)', White Plains.

SALWICKI A. [1970], 'Formalized algorithmic languages', *Bull. Acad. Polon. Sci. , Ser. Sci. Math. Astron. Phys.* **18**, 227–232.

SCHLINGLOFF H. [1992*a*], 'Expressive completeness of temporal logic of trees', *Journal of Applied Non–Classical Logics* **2. 2**, 157–180.

SCHLINGLOFF H. [1992*b*], On the expressive power of modal logic on trees, *in* A. Nerode and M. Taitslin, eds, 'Proc. 2nd Int. Symp. Logical Foundations of Computer Science ("Logic at Tver")', Vol. 620 of *LNCS*, Springer, pp. 441–451.

SCHLINGLOFF H. [1997], 'Verification of finite–state systems with temporal logic model checking', *South African Computer Journal* **19**, 27–52.

SCHLINGLOFF H. AND HEINLE W. [1997], Relational algebra and modal logics, *in* C. Brink, W. Kahl and G. Schmidt, eds, 'Relational Methods in Computer Science', Advances in Computing Science, Springer, chapter 5.

SEGERBERG K. [1968], 'Decidability of S4.1', *Theoria* **34**, 7–20.

SEGERBERG K. [1971], An essay in classical modal logic, Technical Report Filosofiska Studier 13, Department of Philosophy, University of Uppsala.

SISTLA A. P. [1983], Theoretical Issues in the Design and Verification of Distributed Systems, PhD thesis, CMU Dept. of Computer Science, CMU–CS–83–146.

SISTLA A. P. AND CLARKE E. M. [1986], 'Complexity of propositional temporal logics', *Journal of the ACM* **32**(3), 733–749.

SISTLA A. P., VARDI M. Y. AND WOLPER P. [1987], 'The complementation problem for Büchi automata with applications to temporal logic', *Theoretical Computer Science* **49**, 217–237.

SLOAN R. H. AND BUY U. [1997], 'Stubborn sets for real–time Petri nets', *Formal Methods in System Design* **11**(1), 23–40.

STÅLMARCK G. [1989], 'A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula', Swedish Patent No. 467076 (1992), US Patent No. 5 276 897 (1994), European Patent No. 0404 454 (1995).

STÅLMARCK G. AND SÄFLUND M. [1990], Modelling and verifying systems and software in propositional logic, *in* B. K. Daniels, ed., 'Proc. Int. Conf. on Safety of Computer Control Systems (SAFECOMP '90)', Pergamon Press, pp. 31–36.

STARKE P. H. [1991], 'Reachability analysis of Petri nets using symmetries', *Syst. Anal. Model. Simul.* **8**(4/5), 293–303.

STIRLING C. [1987], 'Modal logics for communicating systems', *Theoretical Computer Science* **49**, 311–348.

STIRLING C. [1991], Modal and temporal logics, *in* S. Abramsky, D. Gabbay and T. Maibaum, eds, 'Handbook of Logic in Computer Science', Oxford University Press.

STIRLING C. AND WALKER D. J. [1991], 'Local model checking in the modal μ–calculus', *Theoretical Computer Science* **89**(1), 161–177. also in Proc. TAPSOFT '89, Springer LNCS 351, 369–386, 1989.

TARJAN R. E. [1972], 'Depth first search and linear graph algorithms', *SIAM Journal of Computing* **1**, 146–160.

Tarski A. [1955], 'A lattice–theoretical fixpoint theorem and its applications', *Pacific J. Math.* **5**, 285–309.

Thiagarajan P. S. and Walukiewicz I. [1997], An expressively complete linear time temporal logic for Mazurkiewicz traces, *in* 'Proc. 12th Ann. IEEE Symp. on Logic in Computer Science (LICS '97)', pp. 183–194.

Thomas W. [1990], Automata on infinite objects, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B, Elsevier.

Thomas W. [1999], Languages, automata, and logic, *in* G. Rozenberg and A. Salomaa, eds, 'Handbook of Formal Language Theory', Vol. III, Springer, pp. 389–455.

Touati H. J., Brayton R. K. and Kurshan R. P. [1991], Testing language containment for ω–automata using BDDs, *in* 'Proc. 1991 Int. Workshop on Formal Methods in VLSI Design'.

Valmari A. [1990], A stubborn attack on state explosion, *in* 'Proc. 2nd Workshop on Computer Aided Verification (CAV '90)', Vol. 531 of *LNCS*, Springer, Rutgers, New Brunswick, pp. 156–165.

van Benthem J. [1983], *Modal Logic and Classical Logic*, Bibliopolis, Naples.

van Benthem J. [1984], Correspondence theory, *in* F. G. D. Gabbay, ed., 'Handbook of Philosophical Logic', Reidel, chapter II. 4, pp. 167–249.

van Benthem J. [1991], *The Logic of Time*, 2nd edn, Kluwer, Dordrecht.

Vardi M. [1995], Alternating automata and program verification, *in* J. van Leeuwen, ed., 'Computer Science Today: Recent Trends and Developments', Vol. 1000 of *LNCS*, Springer.

Vardi M. Y. [1982], The complexity of relational query languages, *in* 'Proc. 14th Int. ACM Symp. on the Theory of Computing', pp. 137–146.

Vardi M. Y. and Wolper P. [1986], An automata–theoretic approach to automatic program verification, *in* 'Proc. 1st Symp. on Logic in Computer Science (LICS '86)', Boston, Mass.

Walukiewicz I. [1995], Completeness of Kozen's axiomatisation of the propositional μ–calculus, *in* 'Proc. 10th Ann. IEEE Symp. on Logic in Computer Science (LICS '95)', pp. 14–24.

Winskel G. [1991], 'A note on model checking the modal ν–calculus', *Theoretical Computer Science* **83**, 157–167. also in Proc. ICALP '89, Ausiello et. al. (ed.), LNCS 372, 761–772.

Wolper P. [1982], Specification and synthesis of communicating processes using an extended temporal logic, *in* 'Proc. 9th Int. Symp. on Principles of Programming Languages (POPL '82)', Albuquerque, pp. 20–33.

Wolper P. [1983], 'Temporal logic can be more expressive', *Information and Control* **56**(1–2), 72–99.

Wolper P. [1985], 'The tableau method for temporal logic: An overview', *Logique et Analyse* **110–111**, 119–136.

Wolper P. [1986], Expressing interesting properties of programs in propositional temporal logic, *in* 'Proc. 13th ACM Symp. on Principles of Programming Languages (POPL '86)'.

Wolper P. and Lovinfosse V. [1989], Verifying properties of large sets of processes with network invariants, *in* J. Sifakis, ed., 'Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems', Vol. 407 of *LNCS*, Springer.

Woo T. Y. C. and Lam S. S. [1993], A semantic model for authentication protocols, *in* 'Proc. IEEE Symp. on Research in Security and Privacy'.

Yoneda T., Nakade K. and Tohma Y. [1989], A fast timing verification method based on the independence of units, *in* 'Proc. of 19th Int. Symp. on Fault-tolerant Computing', pp. 134–141.

Yoneda T. and Schlingloff H. [1997], 'Efficient verification of parallel real–time systems', *Formal Methods in System Design* **11**(2), 187–215.

Yovine S. [1997], 'Kronos: A verification tool for real–time systems', *Software Tools for Technology Transfer* **1**(1/2).

Yovine S. [1998], Model–checking timed automata, *in* G. Rozenberg and F. Vandraager, eds, 'Embedded Systems', LNCS, Springer.

Zhang H. [1997], SATO: An efficient propositional prover, *in* 'Proc. Int. Conf. on Automated Deduction (CADE '97)', Vol. 1249 of *LNCS/LNAI*, Springer, pp. 272–275.

List of symbols used in this chapter

Syntactic categories

$\mathcal{P}$ Propositions, Predicates $\{\mathtt{p}, \mathtt{on}, \mathtt{buffer\_empty}...\}$ 1642
$\mathcal{R}$ Relations $\{R, a, b, a_1, ...\}$; $\mathcal{R}^+ \triangleq \mathcal{R} \cup \{\prec, <, =\}$ 1643
$\mathcal{T}$ Time variables $\{t, t_1, ...\}$ 1643
$\mathcal{Q}$ Proposition variables $\{q, p, ...\}$ 1658

Operators

$\bot$, $\rightarrow$ basic boolean operators 1642
$\top$, $\vee$, $\wedge$, $\leftrightarrow$, Ite derived boolean operators 1642
$\hat{\Phi}$ Conjunction $(\varphi_1 \wedge ... \wedge \varphi_n)$ of the finite set $\Phi = \{\varphi_1, ..., \varphi_n\}$ 1690
$\langle R \rangle$ Multi-modal diamond, basic **ML**-operator 1644
$[R]$ Multi-modal box, $[R]\,\varphi \triangleq \neg \langle R \rangle \neg\varphi$ 1644
$\mathbf{X}$ next, $\mathbf{X}\,\varphi \triangleq (\bot\,\mathbf{U}^{+}\,\varphi)$ (i.e., $\mathbf{X} \;\equiv\; \langle\prec\rangle$) 1645
$\cancel{\mathbf{X}}$ weak next, $\cancel{\mathbf{X}}\,\varphi \triangleq \neg\,\mathbf{X}\,\neg\varphi$ (i.e., $\cancel{\mathbf{X}} \;\equiv\; [\prec]$) 1645
$\mathbf{F}^{+}$ sometime, $\mathbf{F}^{+}\,\varphi \triangleq (\top\,\mathbf{U}^{+}\,\varphi)$ (i.e., $\mathbf{F}^{+} \;\equiv\; \langle<\rangle$) 1645
$\mathbf{G}^{+}$ always, $\mathbf{G}^{+}\,\varphi \triangleq \neg\,\mathbf{F}^{+}\,\neg\varphi$ (i.e., $\mathbf{F}^{+} \;\equiv\; [<]$) 1645
$\mathbf{F}^{*}$ reflexive sometime, $\mathbf{F}^{*}\,\varphi \triangleq (\varphi \vee \mathbf{F}^{+}\,\varphi) = (\top\,\mathbf{U}^{*}\,\varphi)$ (i.e., $\mathbf{F}^{+} \;\equiv\; \langle\leq\rangle$) 1645
$\mathbf{G}^{*}$ reflexive always, $\mathbf{G}^{*}\,\varphi \triangleq (\varphi \wedge \mathbf{G}^{+}\,\varphi) = \neg\,\mathbf{F}^{*}\,\neg\varphi$ (i.e., $\mathbf{F}^{+} \;\equiv\; [\leq]$) 1645
$\mathbf{U}^{+}$ linear until, basic **LTL** operator 1646
$\mathbf{U}^{*}$ reflexive until, $(\varphi\,\mathbf{U}^{*}\,\psi) \triangleq (\psi \vee \varphi \wedge (\varphi\,\mathbf{U}^{+}\,\psi))$ 1647
$\mathbf{U}^{-}$ linear since, basic **LTL** operator 1648
$\mathbf{F}^{-}$, $\mathbf{G}^{-}$ temporal past operators 1648
$\mathbf{F}^{\pm}$ temporal existential operator, $\mathbf{F}^{\pm}\,\varphi \triangleq (\mathbf{F}^{-}\,\varphi \vee \mathbf{F}^{*}\,\varphi)$ 1648
$\mathbf{G}^{\pm}$ temporal universal operator, $\mathbf{G}^{\pm}\,\varphi \triangleq \neg\,\mathbf{F}^{\pm}\,\neg\varphi \equiv (\mathbf{G}^{-}\,\varphi \wedge \mathbf{G}^{*}\,\varphi)$ 1648
$\mathbf{A}\,\mathbf{U}^{+}$ forall-until, basic **CTL** operator 1654
$\mathbf{E}\,\mathbf{U}^{+}$ exists-until, basic **CTL** operator 1654
$\mathbf{E}\,\mathbf{X}$ exists-next 1655
$\mathbf{A}\,\cancel{\mathbf{X}}$ forall-next 1655
$\mathbf{E}\,\mathbf{F}^{+}$ exists-future 1654
$\mathbf{A}\,\mathbf{F}^{+}$ forall-future 1654
$\mathbf{E}\,\mathbf{G}^{+}$ exists-globally 1654
$\mathbf{A}\,\mathbf{G}^{+}$ forall-globally 1654
$\mathbf{E}\,\mathbf{F}^{*}$ reflexive exists-future, $\mathbf{E}\,\mathbf{F}^{*}\,\varphi \triangleq (\psi \vee \mathbf{E}\,\mathbf{F}^{+}\,\psi)$ 1654
$\mathbf{A}\,\mathbf{G}^{*}$ reflexive forall-globally, $\mathbf{A}\,\mathbf{G}^{*}\,\varphi \triangleq \varphi \wedge \mathbf{A}\,\mathbf{G}^{+}\,\varphi$ 1654
ν maximization (greatest fixpoint) operator 1660
μ minimization (least fixpoint) operator, $\mu q\ \varphi \triangleq \neg\nu q\neg(\varphi\{q := \neg q\})$ 1660

Logics

PL Propositional logic, e.g., $(\bot \to \mathtt{p})$ 1642
FOL First order logic, e.g., $\exists t(R(t,t') \land \mathtt{p}(t))$ 1643
ML Multi-modal logic, e.g., $\langle R \rangle \, \mathtt{p}$ 1644
LTL Linear temporal logic, e.g., $(\mathtt{p}\,\mathbf{U}^{+}(\mathtt{q}\,\mathbf{U}^{-}\,\mathtt{p}))$ 1648
CTL Computation tree logic, e.g., $\mathbf{A}\,\mathbf{F}^{+}\ \ \mathbf{A}(\mathbf{E}\,\mathbf{X}\,\mathtt{p}\,\mathbf{U}^{+}\,\mathbf{E}\,\mathbf{F}^{+}\,\mathtt{p})$ 1654
ACTL All-fragment of CTL 1679
CTL* Extended CTL, e.g., $\mathbf{E}(\mathbf{G}^{*}\,\mathbf{F}^{+}\,\mathtt{p})$ 1656
qTL Quantified temporal logic, e.g., $\exists q(\mathtt{p}\,\mathbf{U}^{+}\,q)$ 1658
MSOL Monadic second order logic, e.g., $\exists q \forall t(\mathtt{p}(t) \to q(t))$ 1658
μ**TL** Modal μ-calculus, e.g., $\mu q(\mathtt{p} \lor \langle R \rangle \, q)$ 1660

Semantical notions

$\triangleq$ Equal by definition
U Universe of points in time 1642,1643,1644
$\mathcal{I}$ Interpretation of relations and propositions 1642,1643,1644
$\mathbf{v}$ Valuation of logical variables 1643,1658
w Point 1644
$\mathcal{M}$ Model $\mathcal{M} \triangleq (U, \mathcal{I}, \mathbf{v})$ or $\mathcal{M} \triangleq (U, \mathcal{I}, w)$ or $\mathcal{M} \triangleq (S, \mathcal{I}, \mathbf{v})$ 1643,1644,1748
$\mathcal{F}$ Frame $\mathcal{F} \triangleq (U, \mathcal{I})$ 1644
$\mathcal{L}$ Label of a point, $\mathcal{L}(w) \triangleq \{\mathtt{p} \mid w \in \mathcal{I}(\mathtt{p})\}$ 1665
true, **false** Truth values 1642
0, 1 (, 2, ...) (Binary) Domain elements, Constant symbols 1735
$\models$ Validation relation, language containment 1643,1672
$||\!-$ Following relation, semantical consequence 1690
$\vdash$ Derivation relation, syntactical consequence 1691
R Accessibility relation $(R \in \mathcal{R})$ 1643
$\prec$ Transition relation; $\mathcal{I}(\prec) \triangleq \bigcup\{\mathcal{I}(R) | R \in \mathcal{R}\}$ 1643
$<$ Transitive closure; $\mathcal{I}(<) \triangleq \mathcal{I}(\prec)^*$ 1643
$\varphi^{\mathcal{F}}$ Denotation of a formula; $\varphi^{\mathcal{F}} \triangleq \{w \mid (U, \mathcal{I}, w) \models \varphi\}$ 1662
$\sqsubseteq$ Submodel relation 1677
$\stackrel{\rightarrow}{-}$ Simulation relation 1677
$\stackrel{\leftrightarrow}{-}$ Bisimulation relation 1682
$\equiv$ Equivalence 1688
$\equiv_{\mathbf{L}}$ Equivalence with respect to logic $\mathbf{L}$ 1683
$\varphi\{q := \psi\}$ Replacement of free variable q in φ by ψ 1649

Index

A

abstract BDDs 1762
abstraction 1677
abstraction function 1759
acceptance condition 1667
accepting path 1717
accessibility relation 1643
admissible atom 1716
algebra 1747
always operator 1645
ample set 1751
assume-guarantee 1766
asynchronous communication 1725
asynchronous execution 1725
atom 1716
atomic proposition 1642
auto-bisimulation 1687
automaton, ω-automaton 1666
axiom 1690

B

Büchi automaton 1666
BDD 1736
binary decision diagram 1736
binary encoding 1735
bisimulation 1682
buffer overflow 1732

C

cache coherence 1638
canonical model 1692
closed leaf 1707
combinatorial search 1727
completed tableau 1707
completeness 1691
  of multimodal logic 1692
  of transitive closure 1695
compositionality 1765
computation tree logic 1654
connectedness 1664
consequence relations 1689
consistency 1692
correspondence theory 1694
current point 1644

D

decision procedures 1700
  for branching time 1700
  for natural models 1704
  for transitive closure 1702
decision tree 1737
decomposition of properties 1673
deduction theorem 1696
defined language 1665
denotation 1748
depth first search 1718
derivation 1691
derivation rule 1690
deterministic automaton 1668
deterministic model 1678
domain 1727, 1747
domain name 1747

E

Ehrenfeucht-Fraïssé games 1686
elementary equivalence 1683
elementary net 1726
equality 1643
equivalence reduction 1688
eventuality 1700
expressive completeness 1649
expressiveness 1685
extended sub-formula 1697

F

fair path 1715
fair transition system 1666
finitary transition system 1674
finitely maximal set 1697
first order logic 1643
first order variable 1643
fixed point 1661
frame 1644, 1748
functional system 1637
future formula 1651
future operator 1645

G

generated language 1671
generated submodel 1677
generated word 1671
global model checking 1712
global operator 1645

H

Hilbert system 1690
hybrid system 1725

I

image finiteness 1674, 1680

induction rule 1696
initial atom 1716
initial state 1727
initial validity 1665, 1712
interleaving 1725
interpretation 1642, 1747
invariance 1699
inverse image calculation 1713

K

Knaster-Tarski theorem 1661, 1720
Kripke axiom **(K)** 1692
Kripke model 1644

L

labelling 1665
language containment 1676
language, ω-language 1665
linear temporal logic 1648
literal 1642
liveness property 1673
local model checking 1712
local validity 1691
logic 1689
  first order 1643
  fixpoint 1660
  modal 1644
  propositional 1642
  quantified temporal 1658
  second order 1658
  temporal 1646
logical spectrum 1737
loop condition 1710

M

marking of a net 1726
maximal consistent set 1692
maximal path 1654
modal box formula 1678
modal logic 1644
model 1637
  first order 1643
  Kripke 1644
  natural 1650
  relational 1748
  tree 1654
model checking 1637
  for μ-calculus 1720
  for CTL 1713
  for linear time 1716
  for modal logic 1712
  symbolic 1735, 1744, 1746
  under fairness 1715
monotonic formulas 1661
multi-terminal BDDs 1770

N

natural model 1650
necessitation rule **(N)** 1692
nexttime operator 1645

O

open leaf 1707, 1717
ordered tree form 1737

P

p-morphism 1682
Paige-Tarjan algorithm 1689
parallel transition system 1726
partial order method 1751
partition refinement 1688
past formula 1651
past operator 1648
path 1654
persistent set 1754
Petri net 1726
point 1643
positive formula 1662, 1747
pre-state 1708
predicate 1643
predicate transformer 1662
probabilistic systems 1770
program variable 1727
property 1637, 1672
proposition 1642
proposition variable 1658
propositional μ-calculus 1660
propositional formula 1642
propositional logic 1642

Q

quantified temporal logic 1658

R

reachability relation 1643
reactive system 1637
real concurrency 1752
real time system 1769
recurrence set 1666
recursion axiom 1696
regular expression 1666
relation symbol 1747
relation term 1747
representative interleaving 1752

S

safety property 1673
SCC 1710

second order logic ... 1658
security protocols ... 1771
self-fulfilling SCC ... 1718
separation ... 1651
sequence validity ... 1656
Shannon expansion ... 1737
shared variables program ... 1727
signature ... 1747
simulation relation ... 1677
sleep set ... 1751
sometime operator ... 1645
soundness ... 1691
specification ... 1637
    automaton ... 1670
    examples ... 1730
    logical ... 1641
stable partition ... 1688
standard translation ... 1649
state ... 1727
state explosion ... 1638
strongly connected component ... 1710
structure ... 1747
stubborn set ... 1751
stuttering ... 1725
stuttering equivalence ... 1752
stuttering invariance ... 1752
submodel ... 1677
substitution ... 1649, 1660, 1690
successful tableau ... 1707
symbolic model checking 1735, 1736, 1744
    for μ-calculus ... 1746
    for CTL ... 1744
synchronous communication ... 1725
synchronous processing ... 1725
system
    functional ... 1637
    reactive ... 1637

T

tableau ... 1707
Tarjan's SCC algorithm ... 1718
temporal logic ... 1646
terminal point ... 1646
testing ... 1676
theorem proving ... 1765
theory ... 1689
transition relation ... 1643, 1727
transition system ... 1666, 1726
translation between logics ... 1649, 1658
tree model ... 1654
tree validity ... 1656
truth lemma ... 1692
type ... 1747

U

unfolding ... 1707
uniform component ... 1688
univalence axiom **(U)** ... 1694
universal validity ... 1645, 1691, 1712
universe ... 1643
until operator ... 1646

V

variable
    first order ... 1643
    free ... 1649
    individual ... 1747
    program ... 1727
    proposition ... 1658, 1690
    relation ... 1747
variable valuation ... 1643, 1748

W

weak completeness ... 1696
weakly fair transition system ... 1666
word, ω-word ... 1665