Theory of Reactive Systems

Jan Peleska[1]

Centre of Information Technology
University of Bremen
Germany
June 29, 2009

[1] Email:jp@tzi.de

1 Transition Systems and Kripke Structures

The operational semantics of specification formalisms for reactive systems, as well as of computer programs, can be described by means of state transition systems. For the verification of properties of specifications or programs it is useful to extend the notion of transition systems by adding information about the basic properties which are true in each state. This leads to the definition of Kripke structures. The definitions below follow closely [2, pp. 14] and [1].

Definition 1.1 A *State Transition System* is a triple $TS = (S, S_0, R)$, where

- S is the set of *states*
- $S_0 \subseteq S$ is the set of *initial states*
- $R \subseteq S \times S$ is the *transition relation*

□

An *atomic proposition* is a logical proposition which cannot be divided further. Examples are a, $x < y$, but $x < y \wedge a$ is not considered as atomic because it represents the conjunction of a and $x < y$.

Definition 1.2 A *Kripke Structure* $K = (S, S_0, R, L)$ is a state transition system (S, S_0, R) augmented by a set AP of atomic propositions and a function

$$L : S \to 2^{AP}$$

mapping each state s of K to the set of atomic propositions valid in s.

Furthermore it is required that the transition relation R is *total* in the sense that $\forall s \in S : \exists s' \in S : (s, s') \in R$. □

First Order Representations.

Next, we specialise on specification formalisms where the state space can always be defined by a vector of variables, together with their current values. In this context, a state is a mapping from symbols to current values. The mapping is partial, since the visibility of symbols may depend on scope rules. Let $V = \{x_0, x_1, \ldots\}$ be the set of all variable symbols associated with a specification or a program. For each variable $x \in V$, let D_x denote its type (also called *domain*) comprising all possible values x can assume. We require a special element $\top$ to be contained in each D_x, denoting an undefined variable state, such as an arbitrary input value or a stack variable which is still in an undefined state since no assignments to the variable have been performed so far. Let $D = \bigcup_{x \in V} D_x$ the union over all domains of variables from V. A *valuation* is a partial mapping

$$s : V \nrightarrow D$$

which is compatible with the symbol types D_x in the sense that

$$\forall x \in \text{dom } s : s(x) \in D_x$$

In the transition systems and Kripke structures to consider from now on the state space will always be represented by a set of valuation functions. This has a consequence on the atomic propositions to consider: All information that can be obtained from the fact that a system is in state $s : V \not\to D$ is a consequence from the atomic propositions specifying exactly the valuation of each variable in the current state s, that is,

$$x_0 = s(x_0), x_1 = s(x_1), \ldots \qquad (*)$$

Every other atomic proposition, say, $x_0 < x_1$ can be derived from the propositions (*): For example, $x_0 < x_1$ holds in state s if and only if $s(x_0) < s(x_1)$. For the moment, our set of atomic propositions will therefore be

$$AP = \{x = d \mid x \in V \wedge d \in D_x\} \qquad (**)$$

Observe, however, that we will also consider other atomic propositions later on in order to avoid the state explosion that would occur if we enumerated AP from (**) for variables x with large data types, such as 32 and 64 bit integers and floats.

The special nature of the atomic propositions from AP in (**) implies that the mapping L can be easily determined for a Kripke structure as soon as their state space, initial state and transition relation is known: Considering (*) and (**), the atomic propositions valid in some state s are obviously

$$L(s) = \{x = d \mid x \in V \wedge s(x) = d\}$$

Let ϕ a first order logical formula, x a free variable in ϕ and ε and expression. Then $\phi[\varepsilon/x]$ denotes the formula which results from replacement of every free occurrence of x by ε. This term replacement can be applied more than once, which is written $\phi[\varepsilon_0/x_0, \varepsilon_1/x_1, \ldots]$; in which case the replacements are applied from left to right.

Let $s \in S$ a valuation and ϕ a (first order) logical formula with free variables from $V = \{x_0, x_1, \ldots\}$. We say that ϕ *holds in state* s and write $s \models \phi$, if the formula evaluates to true when replacing every free variable x occurring in ϕ by its valuation $s(x)$; that is, $\phi[s(x_0)/x_0, s(x_1)/x_1, \ldots]$ is a tautology.

Based on the replacement concept, the initial state S_0 of a transition system based on variables and valuations can be specified by means of a first order logical formula I, if S_0 coincides with the set of all valuations where I holds, that is,

$$S_0 = \{s : V \not\to D \mid s \models I\}$$

Given S_0 and assuming that S_0 and D are finite, we can always construct such an I by means of

$$I \equiv \bigvee_{s \in S_0} \left(\bigwedge_{x \in V} x = s(x) \right)$$

If the finiteness assumptions do not hold we can write

$$I \equiv \exists s \in S_0 : \forall x \in V : x = s(x)$$

In analogy, we can specify transition relations by means of first order formulas. In contrast to the initial state formula, however, we now have to consider pre- and

post states. Therefore we consider formulas with free variables in V and $V' = \{x' \mid x \in V\}$ and associate unprimed variable symbols x with the prestate and primed variables with the poststate. Let s, s' two valuations and ψ a formula with free variables in V, V'. We say that ψ holds in (s, s') and write $(s, s') \models \psi$ if

$$\psi[s(x_0)/x_0, s(x_1)/x_1, \ldots, s'(x_0)/x'_0, s'(x_1)/x'_1, \ldots]$$

evaluates to true. With this notation a formula T with free variables in V, V' specifies a transition relation $R \subseteq S \times S$ if

$$R = \{(s, s') \in S \times S \mid (s, s') \models T\}$$

Conversely, given transition relation R we can construct a suitable formula T by

$$T \equiv \exists (s, s') \in R : \forall x \in V, x' \in V' : x = s(x) \wedge x' = s'(x)$$

Example 1.3 Consider two parallel processes `P0, P1` acting on global variables `s, c0, c1`. Suppose the processes are executed on a single-core CPU such that each assignment is atomic but the both processes may have to release the CPU between two arbitrary statements.

```
int s  = 0;
int c0 = 0;
int c1 = 0;
```

```
1     P0 {
2      do { s = 0;
3       while ( s == 0 );
4       c0 = 1; // process data
5       c0 = 0;
6      } while (1);
7     }
8
```

```
1     P1 {
2      do { s = 1;
3       while ( s == 1 );
4       c1 = 1; // process data
5       c1 = 0;
6      } while (1);
7     }
8
```

To capture the complete state space, we add two program counters p_0, p_1 in range $\{1, 2, \ldots, 7\}$ indicating the next statement to be executed by `P0, P1`, respectively. The semantics of this little parallel program is specified as follows: The symbol set of the parallel system is $V = \{p_0, p_1, s, c_0, c_1\}$ with $p_0, p_1 \in \{1, 2, \ldots, 7\}$, $c_0, c_1, s \in \mathbb{B}$. The initial state is captured by the formula

$$I \equiv p_0 = 1 \wedge p_1 = 1 \wedge s = 0 \wedge c_0 = 0 \wedge c_1 = 0$$

The transition relation is specified by the formula

$$
\begin{aligned}
T \equiv\ & (p_0 = 1 \wedge p_0' = 2 \wedge p_1' = p_1 \wedge s' = s \wedge c_0' = c_0 \wedge c_1' = c_1) \vee \\
& (p_0 = 2 \wedge p_0' = 3 \wedge p_1' = p_1 \wedge s' = 0 \wedge c_0' = c_0 \wedge c_1' = c_1) \vee \\
& (p_0 = 3 \wedge s = 0 \wedge p_0' = 3 \wedge p_1' = p_1 \wedge s' = s \wedge c_0' = c_0 \wedge c_1' = c_1) \vee \\
& (p_0 = 3 \wedge s \neq 0 \wedge p_0' = 4 \wedge p_1' = p_1 \wedge s' = s \wedge c_0' = c_0 \wedge c_1' = c_1) \vee \\
& (p_0 = 4 \wedge p_0' = 5 \wedge p_1' = p_1 \wedge s' = s \wedge c_0' = 1 \wedge c_1' = c_1) \vee \\
& (p_0 = 5 \wedge p_0' = 6 \wedge p_1' = p_1 \wedge s' = s \wedge c_0' = 0 \wedge c_1' = c_1) \vee \\
& (p_0 = 6 \wedge p_0' = 2 \wedge p_1' = p_1 \wedge s' = s \wedge c_0' = c_0 \wedge c_1' = c_1) \vee \\
& (p_1 = 1 \wedge p_1' = 2 \wedge p_0' = p_0 \wedge s' = s \wedge c_1' = c_1 \wedge c_0' = c_0) \vee \\
& (p_1 = 2 \wedge p_1' = 3 \wedge p_0' = p_0 \wedge s' = 1 \wedge c_1' = c_1 \wedge c_0' = c_0) \vee \\
& (p_1 = 3 \wedge s = 1 \wedge p_1' = 3 \wedge p_0' = p_0 \wedge s' = s \wedge c_1' = c_1 \wedge c_0' = c_0) \vee \\
& (p_1 = 3 \wedge s \neq 1 \wedge p_1' = 4 \wedge p_0' = p_0 \wedge s' = s \wedge c_1' = c_1 \wedge c_0' = c_0) \vee \\
& (p_1 = 4 \wedge p_1' = 5 \wedge p_0' = p_0 \wedge s' = s \wedge c_1' = 1 \wedge c_0' = c_0) \vee \\
& (p_1 = 5 \wedge p_1' = 6 \wedge p_0' = p_0 \wedge s' = s \wedge c_1' = 0 \wedge c_0' = c_0) \vee \\
& (p_1 = 6 \wedge p_1' = 2 \wedge p_0' = p_0 \wedge s' = s \wedge c_1' = c_1 \wedge c_0' = c_0)
\end{aligned}
$$

For representing the associated Kripke structure we use the encoding $\boxed{\pi_0, \pi_1, \sigma, \zeta_0, \zeta_1}$ for a Kripke state s where $L(s) = \{p_0 = \pi_0, p_1 = \pi_1, s = \sigma, c_0 = \zeta_0, c_1 = \zeta_1\}$. For unfolding the Kripke structure from the specification of the transition system we proceed as follows:

(i) **Construct the initial states:** This is done by finding all solutions $s : V \nrightarrow D$ of the formula I describing the initial state. In our example this is trivial since I specifies exactly one admissible initial value for each variable, so S_0 consists just of the one valuation $s_0 = \{p_0 \mapsto 1, p_1 \mapsto 1, s \mapsto 0, c_0 \mapsto 0, c_1 \mapsto 0\}$. In the general case the set of all valuations s with $s \models I$ has to be constructed. Each initial state s is labelled as described above by $L(s) = \{x_0 = s(x_0), x_1 = s(x_1), \ldots\}$. If the number of variables involved and their data ranges are small this can be done using truth tables for I. For more complex applications more sophisticated methods will be introduced later on.

(ii) **Expand from the initial states:** Starting with each initial state, expand the Kripke structure by applying the transition relation. This process stops as soon as the expansions of all states generated so far have already been generated before. More formally, given a state s which has already reached by the expansion, we need to construct all solutions of $T[s(x_0)/x_0, s(x_1)/x_1, \ldots]$, that is T, with all prestate variables replaced by their actual values in s. Every solution s' gives rise to a new Kripke state with $L(s') = \{x_0 = s'(x_0), x_1 = s'(x_1), \ldots\}$.

Lets expand our initial state 1,1,0,0,0 : Replacing the prestate variables in T with these values results in formula

$$\begin{aligned} T[1/p_0, 1/p_1, 0/s, 0/c_0, 0/c_1] \equiv \\ (p_0' = 2 \wedge p_1' = 1 \wedge s' = 0 \wedge c_0' = 0 \wedge c_1' = 0) \vee \\ (p_1' = 2 \wedge p_0' = 1 \wedge s' = 0 \wedge c_1' = 0 \wedge c_0' = 0) \end{aligned}$$

so initial state 1,1,0,0,0 expands to 2,1,0,0,0 and 1,2,0,0,0 . The resulting complete Kripke structure for the two interacting processes in this example is shown in Fig. 2. Observe that we can also represent the Kripke structure as an infinite tree which is called the *computation tree*. □

Unwinding the Computation Tree.

The following algorithm formalises an unwinding procedure for a finite section of the computation tree associated with a Kripke structure, as illustrated in Example 1.3. Since a state s may occur in more than one place of the computation tree we use tree nodes $N = S \times 2^{AP} \times \mathbb{N}$: $(s, P, n) \in N$ denotes a state $s \in S$ which is inserted as a tree node at level n and has valid atomic propositions $P = L(s)$. The computation tree to be constructed is a structure $TC = (N, \rho, \text{succ}, \text{pred})$ with

- $\rho \in N$ the *root* of the tree
- $\text{succ} : N \to \mathbb{P}(N)$ the successor function mapping each tree node to the set of its children. If $\text{succ}(z) = \varnothing$ then z is called a *leaf* of the tree.
- $\text{pred} : N \to N \cup \{\bot\}$ the predecessor function mapping each node to its parent or – in case of the root node – to $\bot$

The algorithm is shown in Fig. 1. It unwinds the computation tree in a manner where a node becomes a leaf if it already occurs elsewhere *on the same path* on a higher level closer to the root. This representation is interesting in the context of test automation (to be discussed in later chapters) and suffices as a simplified model to prove or disprove assertions about the model with are of a certain restricted nature, to be discussed in the next section.

Exercise. 1. Consider the specification model of component C in Fig. 3. C inputs $x \in \{0, 1, 2\}$ and outputs to $y \in \{-1, 0, 1, 2, \ldots\}$. Its behaviour is modelled in Statechart style: The rounded corner boxes denote *locations*, also called *control states*. Arrows between locations denote *transitions*; a transition arrow without source location marks the initial control state.

Expressions in brackets (like `[x > y]`) specify *guard conditions*: The transition from location `l0` to `l1` can only be taken if `x > y` holds, which means, that the current valuation $s : V \not\to D$ results in $s(x) > s(y)$.

Expressions after a dash, like `/ y = -1;`, denote *actions*, that is, assignments to internal variables (if any) or outputs. An action is executed if its associated transition is taken.

function *computationTree*(**in** (S, S_0, R, L) : KripkeStructure) : $(N, \rho, \text{succ}, \text{pred})$
begin
  $n := 1$; $M := \{(s, L(s), n) \mid s \in S_0\}$; $N := \{\rho\} \cup M$;
  $\text{succ} := \{\rho \mapsto S_0\} \cup \{s \mapsto \varnothing \mid s \in S_0\}$;
  $\text{pred} := \{(s, L(s), n) \mapsto \rho \mid s \in S_0\} \cup \{\rho \mapsto \bot\}$
  while $M \neq \varnothing$ **do**
    $M' := \varnothing$;
    foreach $(s, L(s), n) \in M$ **do**
      foreach $s' \in S$ **do**
        if $(s, s') \in R$ **then**
          $N := N \cup \{(s', L(s'), n+1)\}$;
          $\text{succ}(s, L(s), n) := \text{succ}(s, L(s), n) \cup \{(s', L(s'), n+1)\}$;
          $\text{succ}(s', L(s'), n+1) := \varnothing$;
          $\text{pred}(s', L(s'), n+1) := (s, L(s), n)$;
          if $(\forall k \in \{1, \ldots, n\} : pr_1(\text{pred}^k(s', L(s'), n+1)) \neq s')$ **then**
            $M' := M' \cup \{(s', L(s'), n+1)\}$
          endif
        endif
      enddo
    enddo
    $M := M'$
    $n := n + 1$;
  enddo
  computationTree $:= (N, \rho, \text{succ}, \text{pred})$;
end

Fig. 1. Algorithm for generating a finite portion of the computation tree associated with a Kripke Structure (S, S_0, R, L).

Applying the informal description of the behaviour of C in Example 1.3, specify the initial state and the transition relation as logical formulas. □

Exercise. 2. Following the algorithm described in Fig. 1, draw the initial part of the computation tree representing associated with the Kripke structure of C in Exercise 1. For the first 3 nodes in the tree, explain how they are derived from the transition relation. For this exercise assume `N = 2`. □

2 Property Specification With Temporal Logic

*2.1 The Computation Tree Logic CTL**

Operators.

CTL* formulas are based on the following operators:

- The *path quantifiers* are

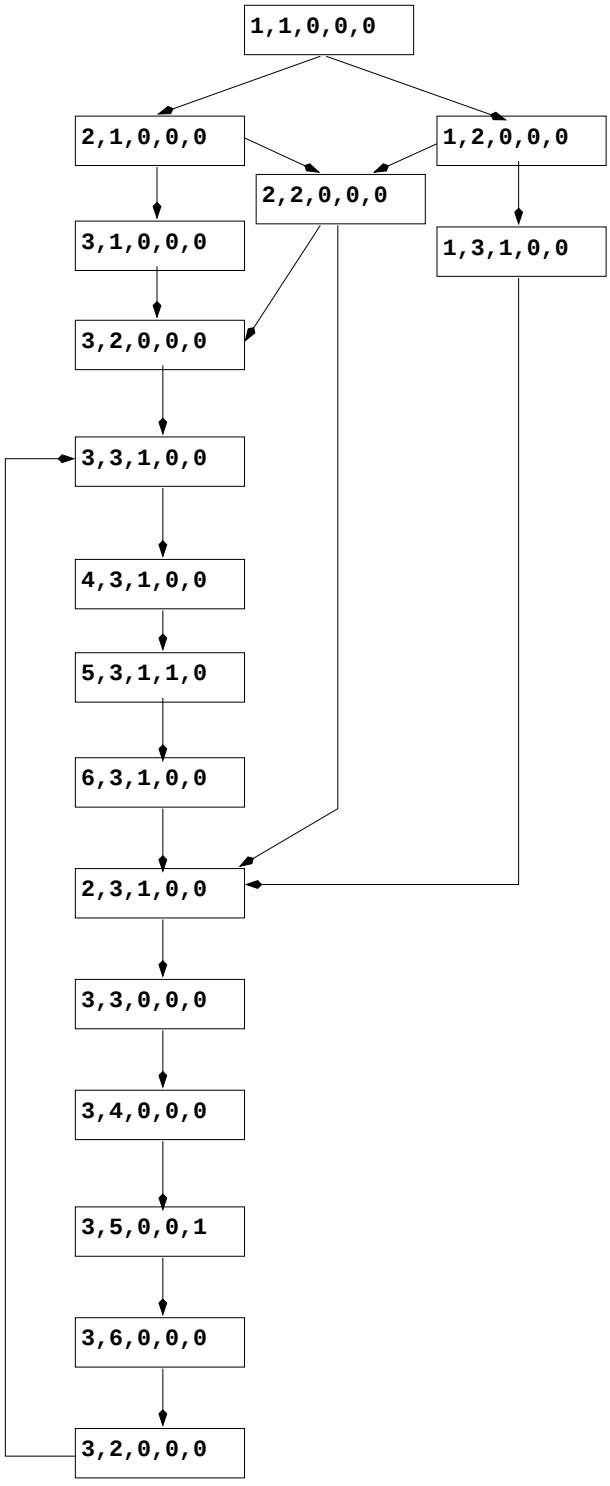
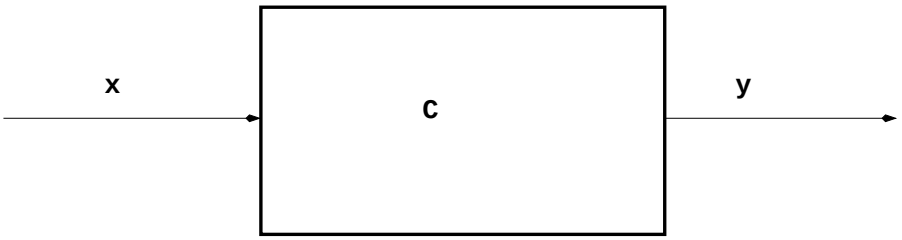


Fig. 2. Kripke structure for the processes $P_0 \parallel P_1$ from Example 1.3.

  - **A** ("on every path")
  - **E** ("there exists a path")
- The *temporal operators* are
  - **X** ("next time")



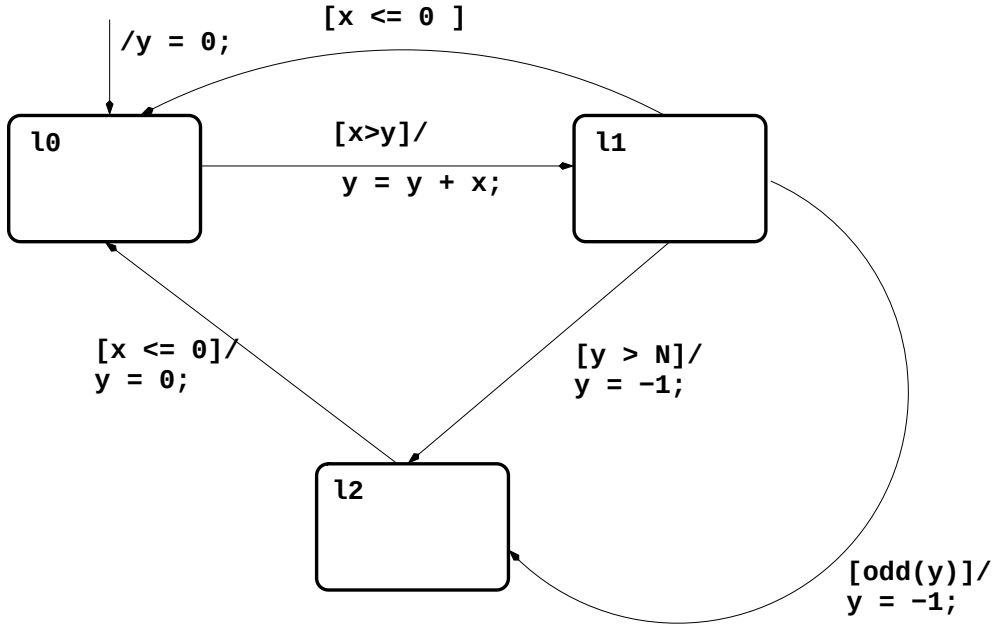


Fig. 3. Model of component C.

- **G** ("globally" or "always")
- **F** ("eventually" or "finally")
- **U** ("until")
- **R** ("release")

Apart from these new operators the conventional Boolean operators can be used, as will be specified in the syntax definition below.

Syntax of CTL* formulas.

CTL* distinguishes between

- *state formulas* which refer to properties of a specific Kripke state
- *path formulas* which specify properties of a path in the computation tree.

State and path formulas refer recursively to each other. The set of all valid CTL* formulas is given by the *state* formulas generated according to the following inductive rules:

(i) Every atomic proposition $p \in AP$ is a state formula.

(ii) If f and g are state formulas then $\neg f, f \wedge g, f \vee g$ are state formulas.

(iii) If f is a *path formula* then $\mathbf{E}\ f, \mathbf{A}\ f$ are *state formulas*.

The path formulas are defined according to the following rules:

(iv) Every state formula is also a path formula.

(v) If f and g are path formulas, then $\neg f, f \wedge g, f \vee g$ are path formulas.

(vi) If f and g are path formulas, then $\mathbf{X}\ f, \mathbf{F}\ f, \mathbf{G}\ f, f\ \mathbf{U}\ g, f\ \mathbf{R}\ g$ are path formulas.

More formally, we can write these syntax rules in EBNF notation as follows, where $p \in AP$, ϕ denotes state formulas and ψ denotes path formulas

$$\text{CTL}^*\text{-formula} ::= \phi$$

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{E}\ \psi \mid \mathbf{A}\ \psi$$

$$\psi ::= \phi \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\ \psi \mid \mathbf{F}\ \psi \mid \mathbf{G}\ \psi \mid \psi\ \mathbf{U}\ \psi \mid \psi\ \mathbf{R}\ \psi$$

Semantics of CTL* formulas.

The semantics of CTL* formulas is explained using a Kripke structure M, specific states s of M and paths π through the computation tree of M. We write

$$M, s \models \phi\ (\phi \text{ a state formula})$$

to express that ϕ holds in state s of M. We write

$$M, \pi \models \psi (\psi \text{ a path formula})$$

to express that ψ holds along path π through M. For CTL* formulas ϕ we say ϕ *holds in the Kripke model* M and write

$$M \models \phi$$

if and only if $\forall s_0 \in S_0 : M, s_0 \models \phi$. For paths $\pi = s_0 s_1 s_2 \ldots$ $\pi(i)$ denotes the ith element s_i of π, and $\pi^i = s_i s_{i+1} \ldots$ the ith suffix of π.

The inductive definition of $\models$ is given in Fig. 4, where p denotes atomic propositions from AP, ϕ, ϕ_i denote state formulas and ψ, ψ_j denote path formulas:

Exercise. 3. Using the syntax rules of CTL* formulas and a syntax tree representation, prove or disprove that the following formulas conform to the CTL*-syntax ($a, b, c \in AP$):

(i) $\mathbf{AG}(\mathbf{XF}a \wedge \neg(b\mathbf{UG}c))$

(ii) $\mathbf{AXG}\neg a \wedge \mathbf{EFG}(a \vee \mathbf{A}(b\mathbf{U}a))$

□

$$
\begin{array}{ll}
M, s \models p & \equiv p \in L(s) \\
M, s \models \neg\phi & \equiv M, s \not\models \phi \\
M, s \models \phi_1 \vee \phi_2 & \equiv M, s \models \phi_1 \text{ or } M, s \models \phi_2 \\
M, s \models \phi_1 \wedge \phi_2 & \equiv M, s \models \phi_1 \text{ and } M, s \models \phi_2 \\
M, s \models \mathbf{E}\ \psi & \equiv \text{there is a path } \pi \text{ from } s \text{ such that } M, \pi \models \psi \\
M, s \models \mathbf{A}\ \psi & \equiv \text{on every path } \pi \text{ from } s \text{ holds } M, \pi \models \psi \\
M, \pi \models \phi & \equiv M, \pi(0) \models \phi \\
M, \pi \models \neg\psi & \equiv M, \psi \not\models \psi \\
M, \pi \models \psi_1 \vee \psi_2 & \equiv M, \pi \models \psi_1 \text{ or } M, \pi \models \psi_2 \\
M, \pi \models \psi_1 \wedge \psi_2 & \equiv M, \pi \models \psi_1 \text{ and } M, \pi \models \psi_2 \\
M, \pi \models \mathbf{X}\ \psi & \equiv M, \pi^1 \models \psi \\
M, \pi \models \mathbf{F}\ \psi & \equiv \text{there exists } k \geq 0 \text{ such that } M, \pi^k \models \psi \\
M, \pi \models \mathbf{G}\ \psi & \equiv \text{For all } k \geq 0\ M, \pi^k \models \psi \\
M, \pi \models \psi_1 \mathbf{U} \psi_2 & \equiv \text{there exists } k \geq 0 \text{ such that } M, \pi^k \models \psi_2 \text{ and for all } 0 \leq j < k\ M, \pi^k \models \psi_1 \\
M, \pi \models \psi_1 \mathbf{R} \psi_2 & \equiv \text{for all } j \geq 0 \text{ holds: if } M, \pi^i \not\models \psi_1 \text{ for every } i < j \text{ then } M, \pi^j \models \psi_2
\end{array}
$$

Fig. 4. Semantics of CTL* formulas.

Exercise. 4. Using the Kripke structure displayed in Fig. 2 prove or disprove the following CTL*-assertions, using the semantic definition described in Fig. 4 in a step-by step manner. For each of the formulas, give a textual interpretation of their meaning.

(i) $\mathbf{AG}\neg(c_0 \wedge c_1)$

(ii) $\mathbf{A}(\mathbf{F}c_0 \wedge \mathbf{G}(c_0 \Rightarrow \mathbf{F}(c_1 \wedge \mathbf{F}c_0)))$

Justify why the first assertion could be proved on the finite representation of the Kripke structure's computation tree as explained in algorithm 1 while this is not possible for the second assertion. □

2.2 The Computation Tree Logic CTL

A frequently used subset of CTL* is called CTL. It is defined by the following restricted syntactic rule (CTL.iv) for the path formulas (the other rules (i), (ii), (iii), (iv), (v) for CTL* syntax applay in the same way to CTL):

(CTL.vi) If f and g are *state formulas* then $\mathbf{X}\ f, \mathbf{F}\ f, \mathbf{G}\ f, f\ \mathbf{U}\ g, f\ \mathbf{R}\ g$ are path formulas.

More formally, the CTL syntax is defined by (p denotes atomic propositions from AP)

$$\text{CTL-formula} ::= \phi$$

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{E}\ \psi \mid \mathbf{A}\ \psi$$

$$\psi ::= \phi \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\ \phi \mid \mathbf{F}\ \phi \mid \mathbf{G}\ \phi \mid \phi\ \mathbf{U}\ \phi \mid \phi\ \mathbf{R}\ \phi$$

As a consequence, the temporal operators $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}, \mathbf{R}$ can never be prefixed by another temporal operator in CTL. Only pairs consisting of path quantifier and temporal operator can occur in a row.

Example 2.1 The CTL* formula $\mathbf{A}(\mathbf{FG}f)$ (*On every path, f will finally hold in all states*) has no equivalent in CTL. □

Theorem 2.2 *Every CTL formula can be expressed by means of the operators* $\neg, \vee, \mathbf{EX}, \mathbf{EU}, \mathbf{EG}$.

Proof. Obviously $\psi_1 \wedge \psi_2$ can be expressed as $\neg(\neg\psi_1 \vee \neg\psi_2)$. The theorem now follows from the fact that the following equivalences hold for all CTL path formulas ψ, ψ_1, ψ_2:

1. $\mathbf{AX}\psi \equiv \neg\mathbf{EX}(\neg\psi)$
2. $\mathbf{EF}\psi \equiv \mathbf{E}(\mathtt{true}\mathbf{U}\psi)$
3. $\mathbf{AG}\psi \equiv \neg\mathbf{EF}(\neg\psi)$
4. $\mathbf{AF}\psi \equiv \neg\mathbf{EG}(\neg\psi)$
5. $\mathbf{A}(\psi_1\mathbf{U}\psi_2) \equiv \neg\mathbf{E}(\neg\psi_2\mathbf{U}(\neg\psi_1 \wedge \neg\psi_2)) \wedge \neg\mathbf{EG}\neg\psi_2$
6. $\mathbf{A}(\psi_1\mathbf{R}\psi_2) \equiv \neg\mathbf{E}(\neg\psi_1\mathbf{U}\neg\psi_2)$
7. $\mathbf{E}(\psi_1\mathbf{R}\psi_2) \equiv \neg\mathbf{A}(\neg\psi_1\mathbf{U}\neg\psi_2)$
8. $\mathbf{E}\phi \equiv \mathbf{E}(\mathtt{false}\mathbf{U}\phi)$ if ϕ does not contain $\mathbf{E}, \mathbf{A}, \mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{UR}$
9. $\mathbf{A}\phi \equiv \neg\mathbf{E}(\mathtt{false}\mathbf{U}\neg\phi)$ if ϕ does not contain $\mathbf{E}, \mathbf{A}, \mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{UR}$

The proof of these equivalences is performed using the semantic rules given in Fig. 4, to be performed by the reader in Exercise 5. □

Exercise. 5. Prove the 9 semantic equivalences used in the proof of Theorem 2.2.

2.3 The Computation Tree Logics ACTL and ACTL*

If we restrict CTL* formulas to universal quantification only, the resulting computation tree logic is called ACTL*. More precisely, ACTL* only admits CTL* formulas satisfying

- The formula is in *positive normal form*, that is, the negation operator $\neg$ is only applied to atomic propositions.
- The only occurring path quantifier is $\mathbf{A}$.

The corresponding restriction of CTL formulas to universal quantification is called ACTL.

Example 2.3 **AFAX**a is an ACTL formula, but **AGEF**a is not in ACTL*, since its **E**-free representation **AG**$\neg$**AG**$\neg a$ is not in positive normal form. □

In Section 4.4 we will prove a theorem about simulation relations between Kripke structures, and the properties that may be transferred from an abstract Kripke structure to its associated concrete one. It will turn out that a sufficient condition for this implication from abstract to concrete level is for the formula to be in the subset of ACTL* or ACTL, respectively.

3 CTL Model Checking

Model checking distinguishes between

- *Equivalence checking.* Two models (these are usually given in state transition system or labelled transition system representation) are compared with respect to semantic equivalence.
- *Refinement checking.* Two models are compared by means of a (usually transitive) relation which is weaker than equivalence.
- *Property checking.* A model is checked with respect to *an (implicit) specification*: The specification is given by a logical formula stating some desired property of the model. The model is usually represented as a transition system or as a Kripke structure $K = (S, S_0, R, L)$. The specification is most frequently expressed by a temporal logic formula ϕ; an alternative specification formalisms is *trace logic*.

  In the general case we wish to identify all states $s \in S$ where ϕ holds, i. e., $s \models \phi$. In most practical applications the objective is to prove that ϕ holds in every initial state $s \in S_0$ and in every state which is reachable from some initial state by n-fold application of the transition relation R; this is written $K \models \phi$.

In this section we investigate property checking for Kripke structures against CTL formulas. The technique which is introduced here is called *explicit model checking* because it requires to represent the Kripke structure's state space in an explicit way, so that all the necessary atomic propositions of the form $x = \nu$ can be directly derived from each state's representation. This is the oldest form of model checking which is only applicable if state spaces are sufficiently small to be enumerated explicitly.

The basic idea of the property checking algorithm.

The property checking algorithm introduced formally below is based on the following concept:

- The CTL specification formula is decomposed into its (binary) syntax tree.
- Starting at the leaves of the syntax tree (the leaves represent atomic propositions) the algorithm processes a sequence of sub-formulas ϕ_i in bottom-up manner.
- The goal of each processing step is to annotate all states s statisfying $s \models \phi_i$ with the new sub-formula ϕ_i. To this end, a labelling function $L_\phi : S \to CTL$ is used.

function *checkCTL*(**in** (S, S_0, R, L) : KripkeStructure; **in** ϕ : CTL) : $\mathbb{P}(S)$
begin
  label : $S \to 2^{\text{CTL}}$;
  label := $\{s \mapsto \varnothing \mid s \in S\}$;
  calcLabel$((S, S_0, R, L), \phi, \text{label})$;
  checkCTL := $\{s \in S \mid \phi \in \text{label}(s)\}$;
end

Fig. 5. Main algorithm for CTL property checking against Kripke structures.

- The algorithm stops when the last formula ϕ_i having been processed coincides with the specification ϕ.
- The result of the algorithm is the set $\{s \in S \mid \phi \in L_\phi(s)\}$.

Syntax tree representation of CTL formulas.

From Section 2.2 we know that every CTL formula can be represented by means of the operators $\neg, \vee, \mathbf{EX}, \mathbf{EU}, \mathbf{EG}$ alone. The binary syntax tree representation of such a formula can be defined recursively using the tree notation

- ε: empty tree
- $T(t_0, n, t_1)$: tree with root n and left sub-tree t_0 and right sub-tree t_1.

The recursive syntax tree definition $t(\phi)$ for a given CTL formula ϕ is as follows:

(i) If $\phi \in AP$ then $t(\phi) = T(\varepsilon, \phi, \varepsilon)$.
(ii) If $\phi = \neg\phi_1$ then $t(\phi) = T(\varepsilon, \neg, t(\phi_1))$.
(iii) If $\phi = \phi_0 \vee \phi_1$ then $t(\phi) = T(t(\phi_0), \vee, t(\phi_1))$.
(iv) If $\phi = \mathbf{EX}\phi_1$ then $t(\phi) = T(\varepsilon, \mathbf{EX}, t(\phi_1))$.
(v) If $\phi = \mathbf{E}(\phi_0 \mathbf{U} \phi_1)$ then $t(\phi) = T(t(\phi_0), \mathbf{EU}, t(\phi_1))$ [2].
(vi) If $\phi = \mathbf{EG}\phi_1$ then $t(\phi) = T(\varepsilon, \mathbf{EG}, t(\phi_1))$.

Given a tree representation $t(\phi)$ of a formula ϕ, its leaves (i. e. its atomic propositions) can be extracted by means of the function leaves : Tree $\to 2^{AP}$ by means of the following recursive definition:

(i) $\text{leaves}(T(\varepsilon, \phi, \varepsilon)) = \{\phi\}$
(ii) $\text{leaves}(T(\varepsilon, \neg, t(\phi_1))) = \text{leaves}(t(\phi_1))$
(iii) $\text{leaves}(T(t(\phi_0), \vee, t(\phi_1))) = \text{leaves}(t(\phi_0)) \cup \text{leaves}(t(\phi_1))$
(iv) $\text{leaves}(T(\varepsilon, \mathbf{EX}, t(\phi_1))) = \text{leaves}(t(\phi_1))$
(v) $\text{leaves}(T(t(\phi_0), \mathbf{EU}, t(\phi_1))) = \text{leaves}(t(\phi_0)) \cup \text{leaves}(t(\phi_1))$
(vi) $\text{leaves}(T(\varepsilon, \mathbf{EG}, t(\phi_1))) = \text{leaves}(t(\phi_1))$

[2] We regard **EU** as a binary operator, so that formulas $\mathbf{E}(\phi_0 \mathbf{U} \phi_1)$ could be equivalently written as $(\phi_0(\mathbf{EU})\phi_1)$. As a consequence its tree representation is $T(t(\phi_0), \mathbf{EU}, t(\phi_1))$

```
procedure calcLabel(in (S, S_0, R, L) : KripkeStructure;
                    in φ : CTL;
                    inout label : S → 2^CTL)
begin
  if φ ∈ AP then
    foreach s ∈ S do
      if φ ∈ L(s) then
        label(s) := label(s) ∪ {φ};
      endif
    enddo
  elseif t(φ) = T(ε, ¬, t(φ_1)) then
    calcLabel((S, S_0, R, L), φ_1, label);
    foreach s ∈ S do
      if φ_1 ∉ label(s) then
        label(s) := label(s) ∪ {φ};
      endif
    enddo
  elseif t(φ) = T(t(φ_0), ∨, t(φ_1)) then
    calcLabel((S, S_0, R, L), φ_0, label);
    calcLabel((S, S_0, R, L), φ_1, label);
    foreach s ∈ S do
      if φ_0 ∈ label(s) ∨ φ_1 ∈ label(s) then
        label(s) := label(s) ∪ {φ};
      endif
    enddo
  elseif t(φ) = T(ε, EX, t(φ_1)) then
    calcLabel((S, S_0, R, L), φ_1, label);
    foreach s ∈ S do
      if ∃s' ∈ S : R(s, s') ∧ φ_1 ∈ label(s') then
        label(s) := label(s) ∪ {φ};
      endif
    enddo
  elseif t(φ) = T(t(φ_0), EU, t(φ_1)) then
    calcLabelEU((S, S_0, R, L), φ_0, φ_1, label);
  elseif t(φ) = T(ε, EG, t(φ_1)) then
    calcLabelEG((S, S_0, R, L), φ_1, label);
  endif
end
```

Fig. 6. Label calculation – syntax-driven control algorithm.

```
procedure calcLabelEU(in (S, S_0, R, L) : KripkeStructure;
                      in φ_0 : CTL; in φ_1 : CTL;
                      inout label : S → 2^CTL)
begin
  T := ⟨s ∈ S | φ_1 ∈ label(s)⟩;
  foreach s ∈ T do label(s) := label(s) ∪ {E(φ_0 U φ_1)};
  while T ≠ ⟨ ⟩ do
    s := hd(T);
    T := tail(t);
    foreach u ∈ {v ∈ S | R(v, s)} do
      if E(φ_0 U φ_1) ∉ label(u) ∧ φ_0 ∈ label(u) then
        label(u) := label(u) ∪ {E(φ_0 U φ_1)};
        T := T ⌢ ⟨u⟩;
      endif
    enddo
  enddo
end
```

Fig. 7. Algorithm for labelling states with $\mathbf{E}(\phi_0 \mathbf{U} \phi_1)$ formulas.

```
procedure calcLabelEG(in (S, S_0, R, L) : KripkeStructure;
                      in φ_0 : CTL; in φ_1 : CTL;
                      inout label : S → 2^CTL)
begin
  S' := {s ∈ S | φ_1 ∈ label(s)};
  SCC := {C | C is a nontrivial SCC of S'}
  T := ⟨s | ∃C ∈ SCC : s ∈ C⟩;
  foreach s ∈ T do label(s) := label(s) ∪ {EGφ_1};
  while T ≠ ⟨ ⟩ do
    s := hd(T);
    T := tail(t);
    foreach u ∈ {v ∈ S' | R(v, s)} do
      if EGφ_1 ∉ label(u) then
        label(u) := label(u) ∪ {EGφ_1};
        T := T ⌢ ⟨u⟩;
      endif
    enddo
  enddo
end
```

Fig. 8. Algorithm for labelling states with $\mathbf{EG}\phi_1$ formulas.

4 Data Abstraction

This section deals with state space reduction by means of data abstraction.

4.1 Equivalence Classes and Factorisation of Transition Systems

Let $TS = (S, S_0, R)$ a transition system and $\sim \subseteq S \times S$ an equivalence relation on S, that is,

- $\forall s \in S : s \sim s$ (reflexivity)
- $\forall s, s' \in S : s \sim s' \Rightarrow s' \sim s$ (symmetry)
- $\forall s, s', s'' \in S : s \sim s' \wedge s' \sim s'' \Rightarrow s \sim s''$ (transitivity)

Let $S/_\sim$ denote the set of equivalence classes; each class is written in the form $[s] \in S/_\sim$, $[s] =_{\text{def}} \{u \mid s \sim u\}$. An equivalence relation gives rise to a transition system *factorised by* $\sim$ which is defined by

$$\begin{aligned} TS/_\sim &=_{\text{def}} (S/_\sim, S_0/_\sim, R/_\sim) \\ S_0/_\sim &=_{\text{def}} \{[s_0] \mid s_0 \in S_0\} \\ R/_\sim &=_{\text{def}} \{([s], [s']) \mid \exists u \in [s], u' \in [s'] : R(u, u')\} \end{aligned} \tag{1}$$

4.2 Auxiliary Variables and Associated Equivalence Classes

Let us consider now again only state spaces S whose elements are variable valuations $s : V \not\to D, V = \{x_1, x_2, \ldots\}$. Let $AUX = \{a_1, a_2, \ldots\}$ a set of fresh variables such that $V \cap AUX = \varnothing$. Let $e_i(x_1^i, x_2^i, \ldots)$ expressions associated with each $a_i \in AUX$. For a fixed set of auxiliary variables a_i and expressions e_i, extend valuation functions by

$$\begin{aligned} &s_e : V \cup AUX \not\to D \\ &\text{dom } s_e = \text{dom } s \cup \{a_i \in AUX \mid x_1^i, x_2^i, \ldots \in \text{dom } s\} \\ &s_e|_V = s \text{ that is, } \forall x \in V \cap \text{dom } s_e : s_e(x) = s(x) \\ &\forall a_i \in AUX \cap \text{dom } s_e : s_e(a_i) = e_i(s(x_1^i), s(x_2^i), \ldots) \end{aligned}$$

Observe that the expressions $e_i(x_1^i, x_2^i, \ldots)$ induce a type D_{a_i} on the corresponding auxiliary variables a_i. We denote the transition system extended by the variables from AUX and the extended valuations s_e by $TS_e = (S_e, S_{0e}, R)$. Observe that since the transition relation R of TS does not refer to any $a \in AUX$, TS_e has the same transition relation as TS.

A collection of auxiliary variables induces an equivalence relation $\sim$ on $TS_e = (S_e, S_{oe}, R)$ by defining

$$\forall s, s' \in S : s \sim s' \equiv_{\text{def}} (\forall a \in AUX : s_e(a) = s'_e(a))$$

$TS_e/_\sim$ is called the factorisation of TS by means of the *data abstraction*

$$a_i = e_i(x_1^i, x_2^i, \ldots),\ i = 1, 2, \ldots$$

Observe that, given a valuation $(s : V \not\to D) \in S$, its equivalence class $[s]$ may also be regarded as a valuation function on the variables from AUX by setting

$$\forall a_i \in AUX : [s](a_i) =_{\text{def}} e_i(s(x_1), s(x_2), \ldots)$$

The definition of $\sim$ guarantees that this valuation function is well-defined, since all members $s' \in [s]$ fulfil

$$\forall i : e_i(s(x_1), s(x_2), \ldots) = e_i(s'(x_1), s'(x_2), \ldots)$$

Lemma 4.1 *Suppose that the initial state S_0 is characterised by first-order predicate $\mathcal{I}$ with free variables in $V = \{x_1, x_2, \ldots\}$, and that the transition relation $R \subseteq S \times S$ is characterised by predicate $\mathcal{R}$ with free variables in V and $V' =_{def} \{x'_1, x'_2, \ldots\}$. Then the respective predicates for $TS_e/\sim$ are given by*

$$\mathcal{I}/\sim(a_1, a_2, \ldots) =_{def} \exists \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge \mathcal{I}[\xi_1/x_1, \xi_2/x_2, \ldots] \quad (2)$$

$$\begin{aligned} \mathcal{R}/\sim(a_1, a_2, \ldots, a'_1, a'_2, \ldots) =_{def} \exists \xi_1, \xi_2, \ldots, \xi'_1, \xi'_2, \ldots : \\ \forall i : (a_i = e_i(\xi_1, \xi_2, \ldots) \wedge a'_i = e_i(\xi'_1, \xi'_2, \ldots)) \wedge \\ \mathcal{R}[\xi_1/x_1, \xi_2/x_2, \ldots, \xi'_1/x'_1, \xi'_2/x'_2, \ldots] \end{aligned} \quad (3)$$

Proof. From (1) and the fact that $\mathcal{I}$ characterises S_0 we conclude that

$$S_{0e}/\sim = \{[s_0] : AUX \not\to D \mid s_0 : V \cup AUX \not\to D \wedge \mathcal{I}[s_0(x_1)/x_1, s_0(x_2)/x_2, \ldots]\}$$

Therefore, in order to prove correctness of $\mathcal{I}/\sim$, it has to be shown that

$$\begin{aligned} &\overline{S} =_{\text{def}} \{s_a : AUX \not\to D \mid \mathcal{I}/\sim[s_a(a_1)/a_1, s_a(x_a)/a_2, \ldots]\} = \\ &\{s_a : AUX \not\to D \mid \exists \xi_1, \xi_2, \ldots : (\forall i : s_a(a_i) = e_i(\xi_1, \xi_2, \ldots)) \wedge \mathcal{I}[\xi_1/x_1, \xi_2/x_2, \ldots]\} \end{aligned}$$

equals $S_{0e}/\sim$.

We show first that $S_{0e}/\sim \subseteq \overline{S}$: Let $[s_0] \in S_{0e}/\sim$. Define $\xi_i =_{\text{def}} s_0(x_i), i = 1, 2, \ldots$. Then, because $\mathcal{I}[s_0(x_1)/x_1, s_0(x_2)/x_2, \ldots]$ holds, this implies $\mathcal{I}[\xi_1/x_1, \xi_2/x_2, \ldots]$. Furthermore, $[s_0](a_i) = e_i(s_0(x_1), s_0(x_2), \ldots)$ by definition of $[\cdot]$, so $(\forall i : a_i = e_i(\xi_1, \xi_2, \ldots))$. As a consequence, $\mathcal{I}/\sim[[s_0](a_1)/a_1, [s_0](a_2)/a_2, \ldots]$ holds which shows that $[s_0] \in \overline{S}$.

Now we show $\overline{S} \subseteq S_{0e}/\sim$: Let $s_a \in \overline{S}$, then there exist $\xi_1, \xi_2, \ldots$ such that $(\forall i : s_a(a_i) = e_i(\xi_1, \xi_2, \ldots)) \wedge \mathcal{I}[\xi_1/x_1, \xi_2/x_2, \ldots]$. Now define a valuation $s_0 : V \not\to D$ by $s_0(x_i) =_{\text{def}} \xi_i, i = 1, 2, \ldots$. This s_0 is contained in S_0 and therefore $[s_0] \in S_{0e}/\sim$, since $\mathcal{I}[\xi_1/x_1, \xi_2/x_2, \ldots]$ and therefore $\mathcal{I}[s_0(x_1)/x_1, s_0(x_2)/x_2, \ldots]$ holds. Since $s_a(a_i) = e_i(\xi_1, \xi_2, \ldots) = e_i(s_0(x_1), s_0(x_2), \ldots)$, the construction of s_0 implies $s_a = [s_0]$, so $s_a \in S_{0e}/\sim$, and this shows $\overline{S} \subseteq S_{0e}/\sim$ and proves (2).

For proving (3), recall from (1) that the transition relation of the factorised transition system $TS_e/\sim$ is defined by

$$R/\sim =_{\text{def}} \{([s], [s']) \mid \exists u \in [s], u' \in [s'] : R(u, u')\}$$

We define

$$\overline{R} =_{\text{def}} \{(s_a, s'_a) \mid \mathcal{R}/_\sim[s_a(a_1)/a_1, s_a(a_2)/a_2, \ldots, s'_a(a_1)/a'_1, s'_a(a_2)/a_2, \ldots]\}$$

and show that $R/_\sim$ equals $\overline{R}$.

To show that $R/_\sim \subseteq \overline{R}$, suppose that $([s],[s']) \in R/_\sim$. By definition of $[\cdot]$, $R/_\sim$ and $\mathcal{R}$ there exists $u, u' : V \not\to D$ such that

$$\begin{aligned}\forall i : (e_i(s(x_1), s(x_2), \ldots) = e_i(u(x_1), u(x_2), \ldots) \wedge \\ e_i(s'(x_1), s'(x_2), \ldots) = e_i(u'(x_1), u'(x_2), \ldots)) \wedge \\ \mathcal{R}[u(x_1)/x_1, u(x_2)/x_2, \ldots, u'(x_1)/x'_1, u'(x_2)/x'_2, \ldots]\end{aligned}$$

holds. Setting $\xi_i = u(x_i), \xi'_i = u'(x_i), i = 1, 2, \ldots$ yields

$$\forall i : (a_i = e_i(\xi_1, \xi_2, \ldots) \wedge a'_i = e_i(\xi'_1, \xi'_2, \ldots)) \wedge \mathcal{R}[\xi_1/x_1, \xi_2/x_2, \ldots, \xi'_1/x'_1, \xi'_2/x'_2, \ldots]$$

and, since $e_i(s(x_1), s(x_2), \ldots)$ equals $e_i(\xi_1, \xi_2, \ldots)$ and $e_i(s'(x_1), s'(x_2), \ldots)$ equals $e_i(\xi'_1, \xi'_2, \ldots)$, this implies that

$$\mathcal{R}/_\sim[[s](a_1)/a_1, [s](a_2)/a_2, \ldots, [s'](a_1)/a'_1, [s'](a_2)/a'_2, \ldots]$$

holds. This proves $([s],[s']) \in \overline{R}$.

It remains to show that $\overline{R} \subseteq R/_\sim$. To this end, assume that $(s_a, s'_a) \in \overline{R}$. By definition of $\overline{R}$ and $\mathcal{R}/_\sim$ this implies the existence of $\xi_i, \xi'_i, i = 1, 2, \ldots$ such that

$$\begin{aligned}\forall i : (s_a(a_i) = e_i(\xi_1, \xi_2, \ldots) \wedge s'_a(a'_i) = e_i(\xi'_1, \xi'_2, \ldots)) \wedge \\ \mathcal{R}[\xi_1/x_1, \xi_2/x_2, \ldots, \xi'_1/x'_1, \xi'_2/x'_2, \ldots]\end{aligned}$$

Now define

$$s : V \not\to D; s(x_i) \mapsto \xi_i, \quad s' : V \not\to D; s'(x_i) \mapsto \xi'_i, i = 1, 2, \ldots$$

Then $[s] = s_a$ and $[s'] = s'_a$ and $\mathcal{R}[s(x_1)/x_1, s(x_2)/x_2, \ldots, s'(x_1)/x'_1, s'(x_2)/x'_2, \ldots]$ by construction and this implies $R(s, s')$ and finally yields $([s],[s']) \in R/_\sim$. This shows $(s_a, s'_a) \in R/_\sim$ and completes the proof. □

4.3 Data Abstraction on Kripke Structures

Given a Kripke structure $K = (S, S_0, R, L)$ and a set AUX of auxiliary variables with associated expressions $e_i(x^i_1, x^i_2, \ldots)$ we can extend K to a Kripke structure $K_e =_{\text{def}} (S_e, S_{oe}, R, L_e)$ by defining its set of atomic propositions and the labelling

function as

$$AP_e =_{\text{def}} AP \cup AP_{AUX}$$
$$AP_{AUX} =_{\text{def}} \{a_i = \alpha \mid a_i \in AUX \wedge \alpha \in D_{a_i}\}$$
$$L_e : S_e \to 2^{AP_e}$$
$$L_e(s) = L(s) \cup \{a_i = e_i(s(x_1^i), s(x_2^i), \ldots) \mid a_i \in AUX\}$$

If we now factorise K_e's transition system (S_e, S_{oe}, R) by the equivalence relation $\sim$ introduced by AUX then we can extend the abstracted transition system to a Kripke structure by "forgetting" about the original variables in V and considering only the propositions on abstraction variables of AUX. This is done in the obvious way by defining a labelling function

$$L_e/_\sim : S_e/_\sim \to 2^{AP_{AUX}};\ [s] \mapsto \{a_i = e_i(s(x_1^i), s(x_2^i), \ldots) \mid a_i \in AUX\}$$

Note that $L_e/_\sim$ is well-defined since all members of $[s]$ induce the same valuations for all $a_i \in AUX$. As a consequence

$$K_e/_\sim = (S_e/_\sim, S_{0e}/_\sim, R/_\sim, L_e/_\sim)$$

is a well-defined Kripke structure, and the explicit model checking algorithms introduced in Section 3 can be applied to $K_e/_\sim$, as long as we only consider CTL formulas φ over the auxiliary variables from AUX, without any reference to the variables from V. Such a formula would also be applicable to the unfactorised Kripke structure K_e. Therefore we would like to know when a formula φ proven to be valid in $K_e/_\sim$ is also valid in K_e.

Example 4.2 Consider the Kripke Structure depicted in Fig. 9, which is associated with a specification model of a traffic light controller. As is well known to every law-abiding citizen we always stop our cars on red *and* on yellow. Therefore, if we are only interested in knowing when cars are in a halt-state in front of the traffic light, it makes sense to introduce a Boolean auxiliary variable

$$\texttt{stops} =_{\text{def}} (\texttt{tl} = \texttt{red} \vee \texttt{tl} = \texttt{yellow})$$

Factorisation against the equivalence relation introduced by `stops` leads to the abstracted Kripke structure shown in Fig. 10.

Now suppose we wish to prove that $\mathbf{EF}(\texttt{tl} = \texttt{green})$ holds for the Kripke structure of the original model in Fig. 9. The assertion can be readily expressed on abstract level as $\mathbf{EF}(\neg\texttt{stops})$ which obviously holds on abstract level, since every path in Fig. 10 visits $(\texttt{m1}, \neg\texttt{stops})$. Similarly, the condition $\mathbf{AF}(tl = red \vee tl = yellow)$ can be expressed in an abstract way as $\mathbf{AF}\texttt{stops}$. It is easy to see that it holds on abstract level.

In these special cases, the assertions also hold on concrete level, but this is not always the case: On abstracted level we can also prove the formula $\mathbf{EG}(\texttt{stops})$ which obviously does not hold in the concrete model. Conversely, the concrete model satisfies $\mathbf{AF}(\neg\texttt{stops})$ which is false on abstract level. □

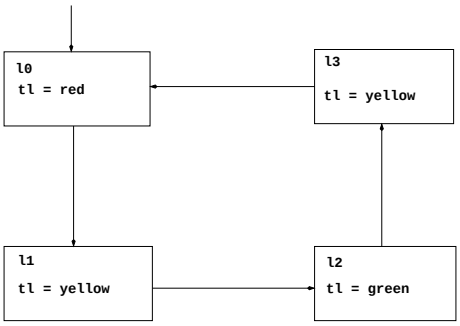


Fig. 9. Kripke structure of traffic light controller from Example 4.2.

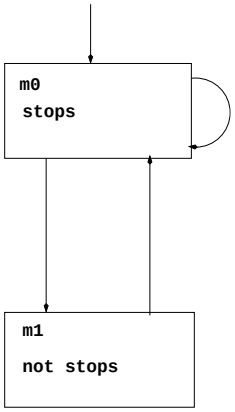


Fig. 10. Abstracted Kripke structure induced by auxiliary variable `stops` in Example 4.2.

Exercise. 6. Consider the slightly modified specification model from Exercise 1, now shown in Fig. 11. Assume now that x and y have unbounded range $D_x = D_y = \mathbb{Z}$, so that explicit model checking becomes infeasible. Chose suitable abstraction variables and construct the corresponding factorisation of the model's Kripke structure such that the following assertion can be proved using the explicit CTL model checking algorithms on the abstracted Kripke structure:

$$\neg\mathbf{EF}(\texttt{l0} \wedge \texttt{odd(y)})$$

Give informal justifications for

- the completeness and correctness of your abstracted Kripke structure (since you do not want to enumerate the concrete (infinite!) Kripke structure of the model),
- the fact that the proof for the abstracted model implies that the assertion also holds for the concrete model.

□

4.4 Simulations

In order to investigate the situations where assertions on auxiliary variables proven on abstract level also hold for the concrete level we introduce the concept of *simulations*:

Definition 4.3 [Simulation] Given two Kripke structures $K = (S, S_0, R, L), K' =$

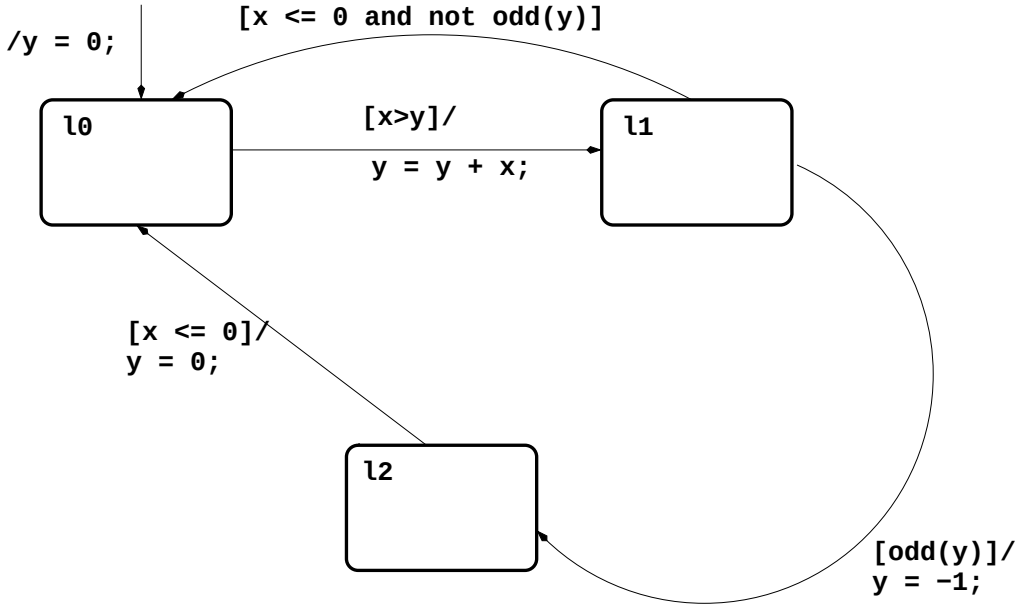


Fig. 11. Model for Exercise 6.

(S', S'_0, R', L') such that K refers to atomic propositions AP and K' refers to atomic propositions AP' and $AP' \subseteq AP$. The relation $H \subseteq S \times S'$ is called a *simulation*, if the following conditions hold for all $(s, s') \in H$:

(i) $L(s) \cap AP' = L'(s')$

(ii) $\forall s_1 \in S : R(s, s_1) \Rightarrow \exists s'_1 \in S' : R'(s', s'_1) \wedge H(s_1, s'_1)$

We write $K \preccurlyeq K'$ *(K is simulated by K')* if such a simulation H exists and

$$\forall s_0 \in S_0 : \exists s'_0 \in S'_0 : H(s_0, s'_0)$$

□

Before exploiting the simulation concept in Theorem 4.7 below it is necessary to show that the equivalence relation $\sim$ induced by auxiliary variables as introduced above establishes a simulation relation between original Kripke structure K_e and its factorisation $K_e/_\sim$:

Theorem 4.4 *Given $\sim$, equivalence classes $[s]$, AP_e, L_e, K_e, $K_e/_\sim$ as introduced in Section 4.3 above, define*

$$H =_{def} \{(s, [s]) \mid s \in S_e\} \subseteq S_e \times S_e/_\sim$$

Then H is a simulation between K_e and $K_e/_\sim$ and $K_e \preccurlyeq K_e/\sim$ holds.

Proof. Let H be defined according to the precondition of the theorem and $s \in S_e$, so that $(s, [s]) \in H$. By the construction rules given in Section 4.3, the states of K_e are labelled with atomic propositions from $AP \cup AP_{AUX}$, and the states (i. e., equivalence classes) of $K_e/_\sim$ are labelled with atomic propositions from AP_{AUX}. As a consequence, the construction of the labelling functions L_e on K_e and $L_e/_\sim$ on

$K_e/\sim$ implies

$$L_e(s) \cap AP_{AUX} = \{a_i = e_i(s(x_1^i), s(x_2^i), \ldots) \mid a_i \in AUX\} = L_e/\sim([s])$$

Therefore condition (i) of Definition 4.3 holds.

Now let $s_1 \in S_e$ such that $R(s, s_1)$. By construction of $R/\sim$ in Section 4.1 this implies $R/\sim([s], [s_1])$ and by construction of H this also implies $H(s_1, [s_1])$. Therefore condition (ii) of Definition 4.3 is also fulfilled.

Finally, we note that $\forall s_0 \in S_0 : H(s_0, [s_0])$ holds by construction of H, and $[s_0] \in S_{0e}/\sim$ by construction of $K_e/\sim$. As a consequence, $K_e \preccurlyeq K_e/\sim$, and this completes the proof. □

Definition 4.5 Let $K \preccurlyeq K'$ with simulation relation $H \subset S \times S'$ and $H(s, s')$. Suppose π is a path in K starting at s and π' a path starting at s' in K'. We say that π *and* π' *correspond to each other* if

$$\forall i \geq 0 : H(\pi(i), \pi'(i))$$

□

Lemma 4.6 *Let $K \preccurlyeq K'$ with simulation relation $H \subset S \times S'$ and $H(s, s')$. Then for every path π in K starting at s there is a corresponding path π' in K' starting at s'.*

Proof. Since π is a path starting at s,

$$\pi(0) = s \wedge (\forall i \geq 0 : R(\pi(i), \pi(i+1)))$$

follows. Since $s = \pi(0)$ and $H(s, s')$, this implies $H(\pi(0), s')$. Applying condition (ii) of Definition 4.3 successively on $\pi(0), \pi(1), \pi(2), \ldots$ this yields the existence of states $\pi'(i) \in S', i \geq 0$, such that

$$\pi'(0) = s' \wedge (\forall i \geq 0 : R'(\pi'(i), \pi'(i+1)) \wedge H(\pi(i+1), \pi'(i+1))),$$

so π' is a path in K', and it corresponds to π by construction. □

Theorem 4.7 *Assume $K \preccurlyeq K'$. Then for every ACTL* formula ϕ with atomic propositions in AP'*

$$(K' \models \phi) \text{ implies } (K \models \phi)$$

Proof. Let ϕ an ACTL* formula as defined in Section 2.3. Suppose $K' \models \phi$, which is equivalent to $\forall s_0' \in S_0' : (K', s_0') \models \phi$. We have to show that for any $s_0 \in S_0$, $(K, s_0) \models \phi$ holds. This is achieved by proving the more general fact that

$$\forall (s, s') \in H : ((K', s') \models \phi) \Rightarrow ((K, s) \models \phi) \qquad (*)$$

which implies our original proof goal. The proof of (*) is performed by structural induction over the formula ϕ. Assume $(s, s') \in H$ and $(K', s') \models \phi$ for the rest of this proof.

(1) If ϕ is an atomic proposition, then $(K, s) \models \phi$ if and only if $\phi \in L(s)$. Since $(K', s') \models \phi$ by assumption, ϕ must be contained in AP'. Because K' simulates K and $L(s) \cap AP' = L'(s')$ holds (condition (i) of Definition 4.3). Now $K' \models \phi$, and therefore $\phi \in L'(s')$ and $L'(s') = L(s) \cap AP'$, so $\phi \in L(s)$ follows.

(2) Let $\phi = \neg\phi_1$ and suppose $(K', s') \models \phi$. Since ϕ is an ACTL* formula ϕ_1 must be an atomic proposition. This implies that $\phi_1 \notin L'(s')$ and, since $L'(s') = L(s) \cap AP'$ and $\phi_1 \in AP'$ also $\phi_1 \notin L(s)$. This means $K, s \not\models \phi_1$ and therefore $K, s \models \neg\phi_1$ which is equivalent to $K, s \models \phi$.

(3) Let $\phi = \phi_1 \vee \phi_2$ such that ϕ_i are state formulas for $i = 1, 2$ and $(K, s) \models \phi_i$ whenever $(K', s') \models \phi_i$. Since $(K', s') \models \phi$, $(K', s') \models \phi_1$ or $(K', s') \models \phi_2$ follows. If $(K', s') \models \phi_1$ then we know already that $(K, s) \models \phi_1$ follows, and this implies $(K, s) \models \phi_1 \vee \phi_2$. The same argument applies if $(K', s') \models \phi_2$. As a consequence $(K, s) \models \phi_1$ or $(K, s) \models \phi_2$ holds, which proves $(K, s) \models \phi_1 \vee \phi_2$.

(4) Let $\phi = \phi_1 \wedge \phi_2$ such that ϕ_i are state formulas for $i = 1, 2$ and $(K, s) \models \phi_i$ whenever $(K', s') \models \phi_i$. This case is handled in analogy to (3).

(5) Let ϕ a state formula, such that $(K, s) \models \phi$ whenever $(K', s') \models \phi$. Let π a path with $\pi(0) = s$, and π' its corresponding path in K', starting at $s' = \pi'(0)$ (this path exists according to Lemma 4.6). Suppose that $K', \pi' \models \phi$ (remember that every state formula is also a path formula). This is equivalent to $K', \pi'(0) \models \phi$, so by our assumption $K, \pi(0) \models \phi$. This implies that $K, \pi \models \phi$. Now we have shown that $K, \pi \models \phi$ whenever $K', \pi' \models \phi$ on a path π' corresponding to π.

(6) Let $\phi = \mathbf{A}\psi$ such that ψ is a path formula and $K, \pi \models \psi$ whenever $K', \pi' \models \psi$, where π, π' are corresponding paths starting in s and s', respectively. Now $K, s \models \mathbf{A}\psi$ is equivalent to the condition that every path π emanating from s satisfies $K, \pi \models \psi$. Since $K', s' \models \mathbf{A}\psi$ we know that $K', \pi'' \models \psi$ for *every* π'' starting at s', so this holds in particular for the path π' corresponding to π. Therefore also $K, \pi \models \psi$ holds, and this implies $K, s \models \mathbf{A}\psi$ since π was an arbitrary path starting at s.

(7) Let $\phi = \psi_1 \vee \psi_2$, such that ψ_i are path formulas where $K, \pi \models \psi_i$ whenever $K', \pi' \models \psi_i$ for $i = 1, 2$ on a path π' corresponding to π. Suppose $K', \pi' \models \psi_1 \vee \psi_2$. This means that $K', \pi' \models \psi_1$ or $K', \pi' \models \psi_2$. By (5) we can deduce that $K, \pi \models \psi_1$ or $K, \pi \models \psi_2$, and we have shown that $K, \pi \models \psi_1 \vee \psi_2$ whenever $K', \pi' \models \psi_1 \vee \psi_2$ on a path π' corresponding to π.

(8) Let $\phi = \psi_1 \wedge \psi_2$, such that ψ_i are path formulas where $K, \pi \models \psi_i$ whenever $K', \pi' \models \psi_i$ for $i = 1, 2$ on a path π' corresponding to π. With an argument analogous to (7) it is shown that $K, \pi \models \psi_1 \wedge \psi_2$ whenever $K', \pi' \models \psi_1 \wedge \psi_2$ on a path π' corresponding to π.

(9) Let $\phi = \mathbf{X}\psi$ and ψ a path formula such that $K, \pi \models \psi$ holds whenever $K', \pi' \models \psi$ holds on a path π' corresponding to π. Now $K', \pi' \models \mathbf{X}\psi$ is equivalent to $K', \pi'^1 \models \psi$. Since π'^1 corresponds to π^1 we know already that $K', \pi'^1 \models \psi$ implies $K, \pi^1 \models \psi$. As a consequence $K, \pi \models \mathbf{X}\psi$ also holds.

(10) The cases $\phi = \mathbf{F}\psi, \phi = \mathbf{G}\psi, \phi = \psi_1\mathbf{U}\psi_2, \phi = \psi_1\mathbf{R}\psi_2$ are shown in analogy to (9), and this completes the proof. □

Theorem 4.8 *Let $K = (S, S_0, R, L)$ and $K' = (S, S_0', R', L)$ Kripke structures with variable symbols from V and atomic propositions AP, using the same set of states S*

and the same labelling function $L : S \to 2^{AP}$. Let $\mathcal{I}$, $\mathcal{I}'$ be the first order predicates characterising the initial states S_0 and S_0', respectively, and $\mathcal{R}$, $\mathcal{R}'$ the first order predicates characterising the transition relations R and R', respectively. Suppose that

- $\mathcal{I} \Rightarrow \mathcal{I}'$
- $\mathcal{R} \Rightarrow \mathcal{R}'$

Then $K \preccurlyeq K'$.

Proof. See Exercise 7. □

Exercise. 7. Prove Theorem 4.8, using the facts on first order representations given in Section 1. □

4.5 Predicate Abstraction

With the knowledge of Section 4.3 alone we could construct abstractions only from the original Kripke structure $K = (S, S_0, R, L)$. This is unsatisfactory, since the very objective of abstraction is to help in situations where the original Kripke structure is too large to be represented in an explicit way. Fortunately there is an alternative for constructing abstractions: Having defined auxiliary variables a_i and associated expressions $a_i = e_i(x_1^i, x_2^i, \ldots)$ we can lift the original predicates $\mathcal{I}, \mathcal{R}$ over $x_j \in V$ specifying initial state and transition relation of K to predicates over a_i specifying initial state and transition relation of the abstracted Kripke structure $K' = (S', S_0', R', L')$. In the next section we will see that this relation can be further approximated by simpler predicates that still preserve the simulation relation but are coarser and therefore even simpler to compute.

Definition 4.9 Let $K = (S, S_0, R, L)$ a Kripke structure with variables from $V = \{x_1, \ldots, x_n\}$ and ϕ a predicate with free variables over V. Let $AUX = \{a_1, \ldots, a_k\}$ a set of auxiliary variables defining an abstraction relation via expressions $a_i = e_i(x_1^i, x_2^i, \ldots), i = 1, \ldots, k$. Then the *lifting of* ϕ with respect to this abstraction is denoted by $[\phi]$ and defined as

$$[\phi] \equiv_{\text{def}} \exists \xi_1, \ldots, \xi_n : (\forall i = 1, \ldots, k : a_i = e_i(\xi_1^i, \ldots, \xi_n^i)) \wedge \phi[\xi_1/x_1, \ldots, \xi_n/x_n]$$

□

Theorem 4.10 *Let $K = (S, S_0, R, L)$ a Kripke structure with variables from $V = \{x_1, \ldots, x_n\}$ and ϕ a predicate with free variables over V. Let $\text{AUX} = \{a_1, \ldots, a_k\}$ a set of auxiliary variables defining an abstraction relation via expressions $a_i = e_i(x_1^i, x_2^i, \ldots), i = 1, \ldots, k$. Let $K' = (S', S_0', R', L')$ denote the abstracted Kripke structure obtained by factorisation with $\sim$ as described in Section 4.3. Let $\mathcal{I}, \mathcal{R}$ denote initial condition and transition relation of K.*

Then initial condition and transition relation of K are given by the lifted predicates

$$[\mathcal{I}] \text{ and } [\mathcal{R}]$$

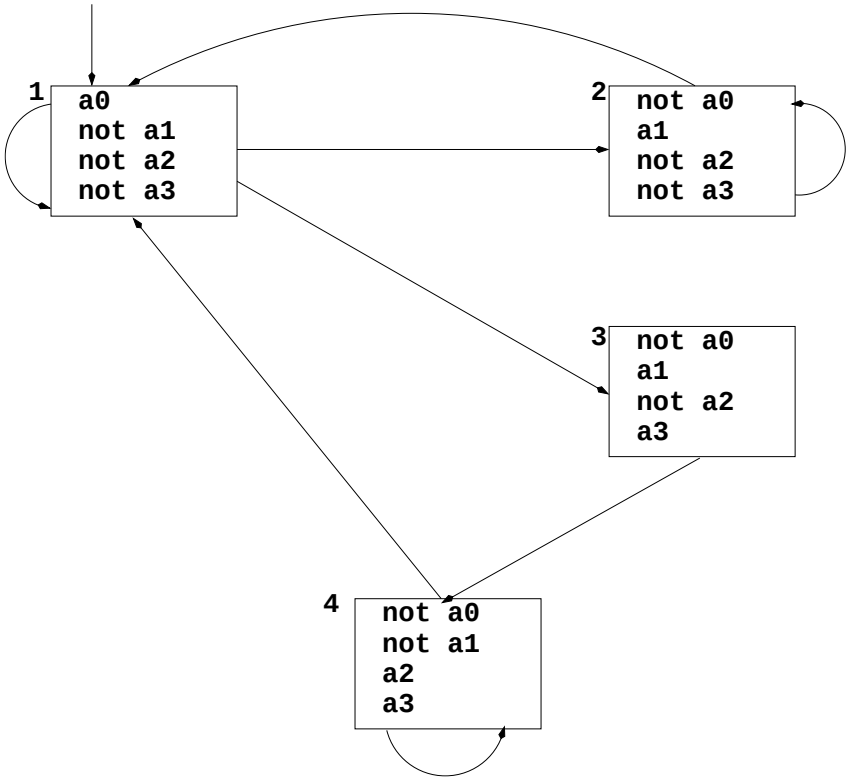


Fig. 12. Kripke structure for abstracted model from Example 4.11.

Proof. Applying Definition 4.9 on $\mathcal{I}$ and $\mathcal{R}$ yields

$$
\begin{aligned}
[\mathcal{I}] &\equiv \exists \xi_1, \ldots, \xi_n : (\forall i = 1, \ldots, k : a_i = e_i(\xi_1, \ldots, \xi_n)) \wedge \mathcal{I}[\xi_1/x_1, \ldots, \xi_n/x_n] \\
[\mathcal{R}] &\equiv \exists \xi_1, \ldots, \xi_n : \exists \xi'_1, \ldots, \xi'_n : (\forall i = 1, \ldots, k : a_i = e_i(\xi_1, \ldots, \xi_n)) \wedge \\
&\qquad (\forall i = 1, \ldots, k : a'_i = e_i(\xi'_1, \ldots, \xi'_n)) \wedge \\
&\qquad \mathcal{R}[\xi_1/x_1, \ldots, \xi_n/x_n, \xi'_1/x'_1, \ldots, \xi'_n/x'_n]
\end{aligned}
$$

According to Lemma 4.1 these formulas represent initial condition $\mathcal{I}/_\sim$ and transition relation $\mathcal{R}/_\sim$ of K'. □

Example 4.11 Consider again the model displayed in Fig. 11 with integer variables x, y having unbounded range. With the knowledge about simulations and predicate abstraction it is now possible to give a rigorous proof for the formula $\neg\mathbf{EF}(\mathtt{l0} \wedge \mathtt{odd(y)})$. First we observe that

$$\neg\mathbf{EF}(\mathtt{l0} \wedge \mathtt{odd(y)}) \equiv \mathbf{AG}(\neg\mathtt{l0} \vee \neg\mathtt{odd(y)})$$

so our proof objective is an ACTL formula. As the simplest abstraction possible for

this objective consider

$$\begin{aligned} a_0 &= \mathtt{l0} \\ a_1 &= \mathtt{l1} \\ a_2 &= \mathtt{l2} \\ a_3 &= \mathtt{odd(y)} \end{aligned} \tag{4}$$

We proceed now to construct the resulting abstracted Kripke structure without first unfolding the one of the concrete system, but exploiting instead its predicates for initial state and transition relation.

Step. 1. Specify initial condition of the concrete system: From Fig. 11 we derive

$$\mathcal{I}(\mathtt{l0}, \mathtt{l1}, \mathtt{l2}, \mathtt{x}, \mathtt{y}) \equiv \mathtt{l0} \wedge \neg \mathtt{l1} \wedge \neg \mathtt{l2} \wedge \mathtt{y} = 0$$

Step. 2. Specify formula for the transition relation of the concrete system: Evaluating Fig. 11 again, we derive

$$\begin{aligned} &\mathcal{R}(\mathtt{l0}, \mathtt{l1}, \mathtt{l2}, \mathtt{x}, \mathtt{y}, \mathtt{l0}', \mathtt{l1}', \mathtt{l2}', \mathtt{x}', \mathtt{y}') \equiv \\ &\quad ((\mathtt{l0} \wedge \mathtt{x} \leq \mathtt{y} \wedge \mathtt{y}' = \mathtt{y} \wedge \mathtt{l0}') \vee \\ &\quad (\mathtt{l0} \wedge \mathtt{x} > \mathtt{y} \wedge \mathtt{y}' = \mathtt{y} + \mathtt{x} \wedge \mathtt{l1}') \vee \\ &\quad (\mathtt{l1} \wedge \mathtt{x} \leq 0 \wedge \neg \mathtt{odd(y)} \wedge \mathtt{y}' = \mathtt{y} \wedge \mathtt{l0}') \vee \\ &\quad (\mathtt{l1} \wedge \mathtt{odd(y)} \wedge \mathtt{y}' = -1 \wedge \mathtt{l2}') \vee \\ &\quad (\mathtt{l1} \wedge \mathtt{x} > 0 \wedge \neg \mathtt{odd(y)} \wedge \mathtt{y}' = \mathtt{y} \wedge \mathtt{l1}') \vee \\ &\quad (\mathtt{l2} \wedge \mathtt{x} \leq 0 \wedge \mathtt{y}' = 0 \wedge \mathtt{l0}') \vee \\ &\quad (\mathtt{l2} \wedge \mathtt{x} > 0 \wedge \mathtt{y}' = \mathtt{y} \wedge \mathtt{l2}')) \wedge \\ &\quad ((\mathtt{l0} \wedge \neg \mathtt{l1} \wedge \neg \mathtt{l2}) \vee (\neg \mathtt{l0} \wedge \mathtt{l1} \wedge \neg \mathtt{l2}) \vee (\neg \mathtt{l0} \wedge \neg \mathtt{l1} \wedge \mathtt{l2})) \wedge \\ &\quad ((\mathtt{l0}' \wedge \neg \mathtt{l1}' \wedge \neg \mathtt{l2}') \vee (\neg \mathtt{l0}' \wedge \mathtt{l1}' \wedge \neg \mathtt{l2}') \vee (\neg \mathtt{l0}' \wedge \neg \mathtt{l1}' \wedge \mathtt{l2}')) \end{aligned}$$

Step. 3. Compute the abstracted initial condition $\mathcal{I}/_\sim = [\mathcal{I}]$: Applying Definition 4.9 on $[\mathcal{I}]$ for the given abstraction (4) results in

$$\begin{aligned} [\mathcal{I}](a_0, a_1, a_2, a_3) \equiv \exists \xi_0, \xi_1, \xi_2, \xi_3, \xi_4 : & \\ a_0 = \xi_0 \wedge a_1 = \xi_1 \wedge a_2 = \xi_2 \wedge a_3 = \mathtt{odd}(\xi_4) \wedge & \\ \xi_0 \wedge \neg \xi_1 \wedge \neg \xi_2 \wedge \xi_4 = 0 & \\ \equiv a_0 \wedge \neg a_1 \wedge \neg a_2 \wedge \neg a_3 & \end{aligned}$$

Step. 3. Compute the abstracted transition relation $\mathcal{R}/_\sim = [\mathcal{R}]$: Applying Defini-

tion 4.9 on $[\mathcal{R}]$ for the given abstraction (4) results in

$$
\begin{aligned}
&[\mathcal{R}](a_0, a_1, a_2, a_3, a_0', a_1', a_2', a_3') \equiv \\
&\quad \exists \xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_0', \xi_1', \xi_2', \xi_3', \xi_4' : \\
&\qquad a_0 = \xi_0 \wedge a_1 = \xi_1 \wedge a_2 = \xi_2 \wedge a_3 = \mathtt{odd}(\xi_4) \wedge \\
&\qquad a_0' = \xi_0' \wedge a_1' = \xi_1' \wedge a_2' = \xi_2' \wedge a_3' = \mathtt{odd}(\xi_4') \wedge \\
&\qquad ((\xi_0 \wedge \xi_3 \leq \xi_4 \wedge \xi_4' = \xi_4 \wedge \xi_0') \vee \\
&\qquad (\xi_0 \wedge \xi_3 > \xi_4 \wedge \xi_4' = \xi_4 + \xi_3 \wedge \xi_1') \vee \\
&\qquad (\xi_1 \wedge \xi_3 \leq 0 \wedge \neg \mathtt{odd}(\xi_4) \wedge \xi_4' = \xi_4 \wedge \xi_0') \vee \\
&\qquad (\xi_1 \wedge \mathtt{odd}(\xi_4) \wedge \xi_4' = -1 \wedge \xi_2') \vee \\
&\qquad (\xi_1 \wedge \xi_3 > 0 \wedge \neg \mathtt{odd}(\xi_4) \wedge \xi_4' = \xi_4 \wedge \xi_1') \vee \\
&\qquad (\xi_2 \wedge \xi_3 \leq 0 \wedge \xi_4' = 0 \wedge \xi_0') \vee \\
&\qquad (\xi_2 \wedge \xi_3 > 0 \wedge \xi_4' = \xi_4 \wedge \xi_2')) \wedge \\
&\qquad ((\xi_0 \wedge \neg \xi_1 \wedge \neg \xi_2) \vee (\neg \xi_0 \wedge \xi_1 \wedge \neg \xi_2) \vee (\neg \xi_0 \wedge \neg \xi_1 \wedge \xi_2)) \wedge \\
&\qquad ((\xi_0' \wedge \neg \xi_1' \wedge \neg \xi_2') \vee (\neg \xi_0' \wedge \xi_1' \wedge \neg \xi_2') \vee (\neg \xi_0' \wedge \neg \xi_1' \wedge \xi_2')) \equiv \\
&\quad ((a_0 \wedge a_3' = a_3 \wedge a_0') \vee (a_0 \wedge a_3' \wedge a_1') \vee (a_0 \wedge \neg a_3' \wedge a_1') \vee \\
&\qquad (a_1 \wedge \neg a_3 \wedge a_3' = a_3 \wedge a_0') \vee (a_1 \wedge \neg a_3 \wedge a_3' = a_3 \wedge a_1') \vee (a_1 \wedge a_3 \wedge a_3' \wedge a_2') \vee \\
&\qquad (a_2 \wedge \neg a_3' \wedge a_0') \vee (a_2 \wedge a_3' = a_3 \wedge a_2')) \wedge \\
&\qquad ((a_0 \wedge \neg a_1 \wedge \neg a_2) \vee (\neg a_0 \wedge a_1 \wedge \neg a_2) \vee (\neg a_0 \wedge \neg a_1 \wedge a_2)) \wedge \\
&\qquad ((a_0' \wedge \neg a_1' \wedge \neg a_2') \vee (\neg a_0' \wedge a_1' \wedge \neg a_2') \vee (\neg a_0' \wedge \neg a_1' \wedge a_2'))
\end{aligned}
$$

The resulting abstracted Kripke structure is displayed in Fig. 12, and it is trivial to see from the graphic representation that $\mathbf{AG}(\neg \mathtt{l0} \vee \neg \mathtt{odd(y)})$ holds, because this formula is equivalent to $\mathbf{AG}(\neg a_0 \vee \neg a_3)$ and the Kripke structure in Fig. 12 simulates the concrete system from Fig. 11 by construction. □

Exercise. 8. Check whether the following C program fragment terminates:

```
1    uint32_t x,y;
2    y = 1;
3    while ( y < 256 ) {
4     x = input(); // Assume 0 <= x <= 15
5     if ( x > y ) {
6       y = y * x;
7     }
8    }
```

Perform this check by means of an abstraction function α that calculates the minimal number of bits needed to represent an integral number:

$$\alpha : \mathbb{N}_0 \to \mathbb{N}_0; \quad x \mapsto \lceil \log_2 x \rceil$$

Observe that, since $\log_b x \cdot y = log_b x + log_b y$, the following estimates hold:

$$\begin{aligned} &\alpha(x \cdot y) \leq \alpha(x) + \alpha(y) \\ &N \leq \alpha(x) + \alpha(y) \Rightarrow N - 1 \leq \alpha(x \cdot y) \\ &\alpha(x) + \alpha(y) \leq N \Rightarrow \alpha(x \cdot y) \leq N \end{aligned}$$

Prove termination or non-termination along the following lines:

(i) Specify initial condition $\mathcal{I}$ and transition formula $\mathcal{R}$ of the concrete program fragment above.

(ii) Now use the abstraction $a_1 = \alpha(x), a_2 = \alpha(y)$. and calculate the abstracted formulas $[\mathcal{I}]$ and $[\mathcal{R}]$.

(iii) Unfold the Kripke structure of the abstracted system given by $[\mathcal{I}]$ and $[\mathcal{R}]$ and sketch how the model checking algorithms introduced in Section 3 come to a conclusion about termination or non-termination.

□

4.6 Predicate Approximation

Depending on the complexity of initial conditions $\mathcal{I}$ and transition relations $\mathcal{R}$ it may be quite hard to compute $[\mathcal{I}]$ and $[\mathcal{R}]$. It is therefore useful to have a technique at hand for further simplifying this computation, at the cost of not arriving exactly at $[\mathcal{I}]$ and $[\mathcal{R}]$, but at *approximations* of these predicates, denoted by $\mathcal{A}(\mathcal{I})$ and $\mathcal{A}(\mathcal{R})$, respectively. We say that predicate ϕ' *approximates* ϕ' if $\phi \Rightarrow \phi'$.

Definition 4.12 Let ϕ a predicate in negation normal form with free variables in $V = \{x_1, x_2, \ldots\}$. Given an abstraction $a_i = e_i(x_1, x_2, \ldots), i = 1, 2, \ldots$, the *approximation of* ϕ is denoted by $\mathcal{A}(\phi)$. $\mathcal{A}(\phi)$ has free variables in $\{a_1, a_2, \ldots\}$ and is defined inductively by the following rules:

(i) If ϕ is an atomic proposition[3], then $\mathcal{A}(\phi) =_{\text{def}} [\phi]$.

(ii) If $\neg\phi$ is a negated atomic proposition, then $\mathcal{A}(\neg\phi) =_{\text{def}} [\neg\phi]$.

(iii) $\mathcal{A}(\phi_1 \wedge \phi_2) =_{\text{def}} \mathcal{A}(\phi_1) \wedge \mathcal{A}(\phi_2)$

(iv) $\mathcal{A}(\phi_1 \vee \phi_2) =_{\text{def}} \mathcal{A}(\phi_1) \vee \mathcal{A}(\phi_2)$

(v) $\mathcal{A}(\exists x : \phi) =_{\text{def}} \exists a : \mathcal{A}(\phi)$

(vi) $\mathcal{A}(\forall x : \phi) =_{\text{def}} \forall a : \mathcal{A}(\phi)$

□

[3] Observe that this includes all primitive relations such as $x < y$, $x = f(y, z)$.

Theorem 4.13 *Let ϕ a predicate in negation normal form with free variables in $V = \{x_1, x_2, \ldots\}$. Given an abstraction $a_i = e_i(x_1, x_2, \ldots), i = 1, 2, \ldots$, the lifted version of ϕ implies its approximated version, i. e.,*

$$[\phi](a_1, a_2, \ldots) \Rightarrow \mathcal{A}(a_1, a_2, \ldots)$$

Proof. The proof is performed by structural induction over the formula ϕ.
Step 1. If ϕ is atomic or the negation of an atom, $\mathcal{A}(\phi) = [\phi]$, so there is nothing to prove.
Step 2. Suppose $\phi \equiv \phi_1 \wedge \phi_2$ and $[\phi_j] \Rightarrow \mathcal{A}(\phi_j),\ j = 1, 2$. From the definition of $[\cdot]$ we calculate

$$\begin{aligned}
[\phi_1 \wedge \phi_2] &\equiv \exists \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge \\
&\qquad \phi_1(\xi_1/x_1, \xi_2/x_2, \ldots) \wedge \phi_2(\xi_1/x_1, \xi_2/x_2, \ldots) \\
&\Rightarrow (\exists \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge \phi_1(\xi_1/x_1, \xi_2/x_2, \ldots)) \wedge \\
&\qquad (\exists \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge \phi_2(\xi_1/x_1, \xi_2/x_2, \ldots)) \\
&\Rightarrow \mathcal{A}(\phi_1) \wedge \mathcal{A}(\phi_2)
\end{aligned}$$

Step 3. Suppose $\phi \equiv \phi_1 \vee \phi_2$ and $[\phi_j] \Rightarrow \mathcal{A}(\phi_j),\ j = 1, 2$. This case is handled in analogy to Step. 2.
Step 4. Suppose $\phi \equiv \exists x : \phi_1$ and $[\phi_1] \Rightarrow \mathcal{A}(\phi_1)$. Assume without loss of generality that $x \neq x_i$ for all $i = 1, 2, \ldots$ and that $\phi = \phi(x, x_1, x_2, \ldots)$. Then

$$\begin{aligned}
[\exists x : \phi_1] &\equiv \exists \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge (\exists \xi : \phi_1(\xi/x, \xi_1/x_1, \xi_2/x_2, \ldots)) \\
&\Rightarrow \exists \xi, \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge \phi_1(\xi/x, \xi_1/x_1, \xi_2/x_2, \ldots) \\
&\Rightarrow \exists \xi : (\exists \xi_1, \xi_2, \ldots : (\forall i : a_i = e_i(\xi_1, \xi_2, \ldots)) \wedge \phi_1(\xi/x, \xi_1/x_1, \xi_2/x_2, \ldots)) \\
&\Rightarrow \exists a : \mathcal{A}(\phi_1)
\end{aligned}$$

Step 5. Suppose $\phi \equiv \forall x : \phi_1$ and $[\phi_1] \Rightarrow \mathcal{A}(\phi_1)$. This step is handled in analogy to Step 4. □

Theorem 4.14 *Given a Kripke structure $K = (S, S_0, R, L)$ with variables in $V = \{x_1, x_2, \ldots\}$, initial condition $\mathcal{I}$ and transition formula $\mathcal{R}$. Given an abstraction $a_i = e_i(x_1, x_2, \ldots), i = 1, 2, \ldots$. Let $K' = (S', S_0', R', L')$ denote the Kripke structure with variables $\{a_1, a_2, \ldots\}$, initial condition $\mathcal{A}(\mathcal{I})$ and transition relation $\mathcal{A}(\mathcal{R})$. Then*

$$K \preccurlyeq K'$$

Proof. Let K'' denote the abstracted Kripke structure with variables $\{a_1, a_2, \ldots\}$, initial condition $[\mathcal{I}]$ and transition formula $[\mathcal{R}]$. From Theorem 4.10 and Theorem 4.4 we know that K'' simulates K. From Theorem 4.13 we know that $[\mathcal{I}] \Rightarrow \mathcal{A}(\mathcal{I})$ and $[\mathcal{R}] \Rightarrow \mathcal{A}(\mathcal{R})$. Now Theorem 4.8 implies that K' simulates K''. Since $\preccurlyeq$ is transitive, the theorem follows. □

Exercise. 9. Given a Kripke structure $K = (S, S_0, R, L)$ we use the following notation:

- $K_s =_{\text{def}} (S, \{s\}, R, L)$ for $s \in S$
- $s_0 \preccurlyeq s_1 \equiv_{\text{def}}$ there exists a simulation relation $H \subseteq S \times S$ such that $H(s_0, s_1)$

Consider the following algorithm:

$H := \{(s_0, s_1) \mid L(s_0) = L(s_1)\};$

while H is not a simulation relation **do**

Choose (s_0, s_1) such that

$\exists s_0' \in S : R(s_0, s_0') \wedge (\forall s_1' \in S : R(s_1, s_1') \Rightarrow (s_0', s_1') \notin H);$

$H := H - \{(s_0, s_1)\};$

enddo

(i) Justify informally why H, as computed by this algorithm, is a simulation relation.

(ii) Explain the relation between H as computed by this algorithm, $s_0 \preccurlyeq s_1$, K_{s_0} and K_{s_1}.

□

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* The MIT Press, 2008.

[2] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* The MIT Press, 1999.