

# Fire-fighting the State-Explosion Problem with Abstraction & Symmetry

Christian Kissig

January 11, 2005

## Abstract

Erroneous system can easily cause costs or even damage on both the customers and the developers side. Therefore it is vital to validate a systems design as early as possible. An effective and thus successful means of design validation is model checking. However, early model-checkers for parallel systems confronted the researchers with the severe problem of state-explosion. In this thesis I will introduce *Abstraction* and *Symmetry* to leverage this problem.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The running Example</b>	<b>2</b>
<b>3</b>	<b>Preliminaries</b>	<b>2</b>
3.1	A Theory of Structures . . . . .	2
3.2	CTL* and ACTL* - Logics for Model Checking . . . . .	5
3.3	Preliminary Results for (A)CTL* and Structures . . . . .	5
3.4	Group Theory . . . . .	5
<b>4</b>	<b>Abstraction</b>	<b>6</b>
4.1	Cone-of-Influence Reduction . . . . .	7
4.2	Data Abstraction . . . . .	10
4.2.1	Approximations . . . . .	10
4.2.2	Constructing Data Abstractions . . . . .	12
4.2.3	Exact Approximations . . . . .	14
<b>5</b>	<b>Symmetry</b>	<b>15</b>
5.1	Theoretical Background . . . . .	15
5.2	Model Checking with Symmetry . . . . .	16
5.3	Complexity Issues . . . . .	17
<b>6</b>	<b>Conclusions</b>	<b>18</b>

# 1 Introduction

*Model Checking* was proved to be an adequate means to validate a systems design up to total security. However, this security comes with severe expenses in terms of computation. The set of states of a system to validate has to contain every possible valuation of each variable occurring in the systems design. This set that we call *state space* can thereby easily become incredibly large. We call this problem the *state explosion problem*.

In this thesis we regard several techniques to reduce the “exploded” state space. All of these techniques have in common that they remove unneeded data from the systems model and thereby reduce the state space. In chapter 4 we regard two abstraction techniques, namely the *cone-of-influence reduction* and *data abstraction*. The first technique filters out correlation between variables, the latter allows us to reduce the state space of each variable. In chapter 5 on symmetry we find out how to exploit symmetries in a systems design.

## 2 The running Example

All the three techniques introduced in this report will be exemplified with the following simple example.

**Example 1 (Crossing)** *As a running example in this report we regard a crossing of two streets. On each street there are two traffic lights - one for each direction - not only for vehicles, but also for pedestrians. Each traffic light changes in the order green, yellow, then red. Opposite traffic lights are set to the same color. If the traffic lights on a street are set to yellow or green then pedestrian signals on the same street are set to red.*

## 3 Preliminaries

In this thesis we will regard model checking in terms of CTL\* (computation tree logic) and its sublogic ACTL\* that we will introduce in 3.2. It is common to define their semantics in terms of Kripke Structures that we will briefly discuss in section 3.1. Structural properties of Kripke Structures seen as Graphs have some interesting consequences when interpreted as models for CTL\* formulae. We will discuss these results in 3.3. For the observations on symmetry we will need some group theory that we will sketch in section 3.4

### 3.1 A Theory of Structures

As stated at the beginning of this section *Kripke Structures* are a means to give semantics to CTL\* formulae. Consisting of a set of states and transitions between them, a Kripke Structure models the dynamic behaviour of a system.

Kripke  
Structures

**Definition 2 (Kripke Structures)** *A Kripke Structure over an alphabet AP of atomic properties is a tuple  $M = \langle S, R \subseteq S \times S, L \rangle$  with  $S$  being the set of states,  $R$  being the next-state relation over  $S$ ,  $S_0 \subseteq S$  the set of initial states, and the labelling  $L$  associating to each state  $s \in S$  a set of atomic properties  $L(s) \subseteq AP$  of the form  $v = d$  for  $v \in V$  a variable and  $d \in D$  a value from the domain  $D$ .*

We write  $s_1 \rightarrow_R s_2$  for  $(s_1, s_2) \in R$ .

Furthermore, we see the systems states as valuations  $V \rightarrow D$  of the variables to a common domain  $D$ . Atomic Propositions are then defined over these valuations and have the form  $v = d$  for  $v \in V$  being a variable and  $d \in D$  a value from domain  $D$ . Similarly, the labelling  $L(s)$  for a state consists of atomic propositions of the same kind.

Notice, that regarding one common domain  $D$  for all variables does not restrict generality. In case the system is modelled with distinguished domains  $D_1, \dots, D_n$  simply take  $D$  to be their disjoint union.

The set  $S$  of states is then just the set of all possible valuations  $V \rightarrow D$ . We assume that the set of initial states  $S_0$  can be distinguished by a first order formula  $\mathcal{S}_0$ , such that  $s \in S_0 \iff \mathcal{S}_0(s)$ . In a similar way the next-state relation  $R$  is determined by a first order formula  $\mathcal{R}$ , such that for every  $s_1, s_2 \in S$  it holds  $s_1 \rightarrow_R s_2 \iff \mathcal{R}(s_1, s_2)$ . In practice these first order representations are manifested in *ordered binary decision diagrams (OBDDs)* for efficiency purposes. However, this speciality does not directly affect our observations, such that we will not explicitly mention OBDDs here.

Structures may be displayed diagrammatically, with vertices (circles) representing the states and arcs representing the connectedness relation  $R$ . The labelling of the states is respresented by attaching the labels  $L(s)$  to a state-vertex. We put circles around vertices to mark the respective states as initial.

In the section on symmetry we will show how to find symmetries in the behaviour of different parts of one Structure. Dividing a Structure into different parts in formalized in the term of a *Substructure*

First Order  
Representa-  
tion

Substructures

**Definition 3 (Substructures)** Let  $M = \langle S, R, S_0, L \rangle$  and  $M' = \langle S', R', S'_0, L' \rangle$  be two Structures. Then we call  $M'$  a Substructure of  $M$  iff the following properties hold

- $S' \subseteq S$
- $S'_0 \subseteq S_0$
- $s \rightarrow_R t$  implies  $s \rightarrow_{R'}$  for any  $s, t \in S'$
- $L'(s) = L(s)$  for any  $s \in S'$

After having defined the technical details of Kripke Structures we are ready to test those definitions on our Crossing Example.

**Example 4 (Crossing - Structure)** Obviously, a state of the crossing is uniquely determined by the valuation of the systems variables  $V = \{t_1, \dots, t_4, p_1, \dots, p_4\}$  to the values in  $D = \{\text{green}, \text{yellow}, \text{red}\}$ . That is we have a state space  $S_{\text{cross}}$  of  $3^8 = 6561$  states. Of cause, this is not very much compared to industrial applications of model-checking. But, it is already too large to be printed readably on a A4 sheet.

We define the transitions  $R_{\text{cross}}$ , i.e. how the traffic lights switch, in terms of a first order formula  $\mathcal{R}_{\text{cross}}$  being defined as follows

$$\begin{aligned} & \mathcal{R}_{\text{cross}}(s_1, s_2) \text{ iff} \\ & (s_1(t_1) = \text{green} \Rightarrow (s_2(t_1) = \text{yellow} \wedge s_2(p_1) = \text{red})) \wedge \\ & (s_1(t_1) = \text{yellow} \Rightarrow (s_2(t_1) = \text{red} \wedge s_2(p_1) = \text{green})) \wedge \end{aligned}$$

$$\begin{aligned}
& (s_1(t_1) = \text{red} \Rightarrow (s_2(t_1) = \text{green} \wedge s_2(p_1) = \text{red})) \wedge \\
& \vdots \\
& (s_1(t_4) = \text{green} \Rightarrow (s_2(t_4) = \text{yellow} \wedge s_2(p_4) = \text{red})) \wedge \\
& (s_1(t_4) = \text{yellow} \Rightarrow (s_2(t_4) = \text{red} \wedge s_2(p_4) = \text{green})) \wedge \\
& (s_1(t_4) = \text{red} \Rightarrow (s_2(t_4) = \text{green} \wedge s_2(p_4) = \text{red}))
\end{aligned}$$

Obviously,  $\mathcal{R}$  is not very secure for pedestrians changing side at the end of the green period. But, this example is not meant to work in practice.

The Crossing is in an initial state if traffic lights on one street are green (pedestrian signals are red there), and traffic lights on the other street are red (pedestrian signals are green). This is expressed by the following FO formula

$$\begin{aligned}
& S_0(s) \text{ iff} \\
& s = \{t_1, t_3 \mapsto \text{green}; p_1, p_3 \mapsto \text{red}, t_2, t_4 \mapsto \text{red}, p_2, p_4 \mapsto \text{green}\} \vee \\
& s = \{t_1, t_3 \mapsto \text{red}; p_1, p_3 \mapsto \text{green}, t_2, t_4 \mapsto \text{green}, p_2, p_4 \mapsto \text{red}\}
\end{aligned}$$

Note, that this choice is arbitrary. There may be other reasonable choices for the initial states.

It is an easy matter to check that no critical state is reachable from the initial states. However, that should be a task for the model checker.

(Bi-) Simulation

Since, Kripke Structures are a special kind of graphs it appears reasonable to define some kind of homomorphism that respects the labelling, namely *Simulation*. Later we will see that Simulation has some nice properties when the underlying Structure is a model of a CTL\* formula enabling us to use Simulation for state space reduction.

**Definition 5 ((Bi-) Simulation between Kripke Structures)** Let  $M = \langle S, R, S_0, L \rangle$  and  $M' = \langle S', R', S'_0, L' \rangle$  be two Structures over the same set of variables  $V$  over domains  $D$  and  $D'$  respectively. Then a relation  $\mathcal{S} \subseteq M \times M'$  is a simulation iff for any pair  $(s, s') \in \mathcal{S}$

1.  $L'(s') = AP' \cap L(s)$
2. for any  $t \in S$  the transition  $s \rightarrow_R t$  implies the existence of some  $t' \in S'$ , such that  $(s', t') \in \mathcal{S}$ .

holds.

If additionally the inverse  $\mathcal{S}^{-1} = \{(t, s) \mid (s, t) \in \mathcal{S}\}$  is a simulation then  $\mathcal{S}$  is called a *Bisimulation*.

A Kripke Structure  $M'$  is said to simulate  $M$ , written  $M \preceq_{\mathcal{S}} M'$  if for any initial state  $s_0 \in S_0$  in  $M$  there is a corresponding initial state  $s'_0 \in S'_0$  in  $M'$ , such that  $(s, s') \in \mathcal{S}$ . If such an  $\mathcal{S}$  furthermore is a bisimulation we say that  $M'$  bisimulates  $M$ , written  $M \simeq_{\mathcal{S}} M'$ .

By applying the definition 5 one can easily prove that  $\preceq_{\mathcal{S}}$  is a partial relation, and  $\simeq_{\mathcal{S}}$  is an equivalence relation, such that for each Structure  $M$  one can construct the equivalence classes  $|s|_{\simeq_{\mathcal{S}}}$  in the canonical way.

**Exercise 6** Prove that  $\preceq_{\mathcal{S}}$  is a partial relation, and  $\simeq_{\mathcal{S}}$  is an equivalence relation. How do the equivalence classes for  $\simeq_{\mathcal{S}}$  look like?

**Exercise 7** Check for our crossing example 4 whether no critical state is reached from the initial ones. For our example critical means no cars crash and no pedestrian is racked in.

### 3.2 CTL\* and ACTL\* - Logics for Model Checking

The behaviour of a System can be understood in terms of a *computation tree* which is nothing but an unwinded structure. The computation tree has nodes representing states, with the root node being the initial state. Descending from each node one can reach nodes representing the connected states. If the structure contains cycles the computation tree has infinite depth.

CTL\*

*Computational Tree Logics (CTL\*)* describe properties of these trees. One distinguishes between *path quantifiers* and *temporal operators*. Path Quantifiers are used to describe the branching behaviour of a system, while Temporal Operators describe the states within a path. In detail we have the following constructors for  $\phi$  and  $\psi$  being CTL\*-formulae.

- Path Quantifiers:  $\mathbf{A}\phi$  (for all paths) and  $\mathbf{E}\phi$  (for some path)
- Temporal Operators :  $\mathbf{X}\phi$  (next state),  $\mathbf{F}\phi$  (eventually),  $\mathbf{G}\phi$  (always)
- Boolean Connectives :  $p \in AP$  (atomic propositions),  $\neg\phi$ ,  $\phi \wedge \psi$ ,  $\phi \vee \psi$
- Composite Connectives :  $\phi\mathbf{U}\psi$  (until),  $\phi\mathbf{R}\psi$  (before)

ACTL\*

In subsequent sections we will need to restrict some results to a sublogic of CTL\*, namely ACTL\*, that is restricted to universal path quantifiers. In detail ACTL\* allows for  $\mathbf{A}$ ,  $\mathbf{X}$ ,  $\mathbf{F}$ ,  $\mathbf{G}$ ,  $\wedge$ ,  $\vee$ ,  $\mathbf{U}$ ,  $\mathbf{R}$ , any for primitive negation  $\neg p$  with  $p \in AP$ , only.

### 3.3 Preliminary Results for (A)CTL\* and Structures

As already mentioned in this preliminary section, Simulation plays an important part when reducing Structures being models of CTL\* formulae. Concretely, Simulation induces preservation of models. The following theorems shall denote this property more accurately.

**Theorem 8** *Simulation induces preservation of ACTL\* models or, formally, assume two structures  $M = \langle S, R, S_0, L \rangle$  and  $M' = \langle S', R', S'_0, L' \rangle$  with a respective set  $AP$  or  $AP'$  of atomic propositions each, such that  $M \preceq M'$ , Then, for every ACTL\* formula  $\phi$  over  $AP'$  it follows that  $M' \models \phi$  implies  $M \models \phi$ .*

If we can establish a bisimulation between those two Structures we can even conclude preservation of CTL\* models in both directions as stated in the following theorem.

**Theorem 9** *Bisimulation induces preservation of CTL\* models. Formally, for two Structures  $M = \langle S, R, S_0, L \rangle$  and  $M' = \langle S', R', S'_0, L' \rangle$  with  $M \simeq M'$ , then for every CTL\* formula  $\phi$  over  $AP'$  it follows that  $M' \models \phi$  iff  $M \models \phi$ .*

Proofs of those theorems will be shown in the presentation on “Equivalences and Preorders between Structures” given by Vu Duc Lam.

### 3.4 Group Theory

Especially in the section on Symmetry we will need some *Group Theory* to formalize the reconfiguration of a Kripke Structure. Therefore we will introduce some basics in advance.

(Sub-) Groups

**Definition 10 ((Sub-) Groups)** A Group is a Set  $|G|$  with a binary operation  $\circ \subseteq X \times X$  (multiplication), such that

1.  $\circ$  is associative, i.e.  $a \circ (b \circ c) = (a \circ b) \circ c$
2. there is an identity element  $e$ , such that  $a \circ e = a = e \circ a$
3. each element  $a$  has an inverse  $a^{-1}$ , such that  $a \circ a^{-1} = e$

A Group  $H$  with the same binary operator  $\circ$  is a Subgroup of  $G$  iff  $|H| \subseteq |G|$ .

Consider a group  $G$  and subset  $X \subseteq |G|$  of the carrier set of  $G$ . Then  $\langle X \rangle$  is the smallest Subgroup of  $G$  containing  $X$ . If a subgroup  $H$  of  $G$  coincides with  $\langle X \rangle$  we call  $H$  generated by  $X$ . Note that in this case  $H$  is the closure of  $X$  under multiplication and inverse.

A bijective Relation  $\sigma$  over a finite Set  $X$  is called a *Permutation*. The set of all Permutations over the Set  $X$  together with functional Composition forms the Group  $SymX$ . The proof of the Group Axioms is easy and therefore postponed to exercise 12.

Generators  
of a (Sub-)  
Group

**Definition 11 (Automorphisms and Invariance Permutations)** Let  $M = \langle AP, S, R, L \rangle$  be a Structure, then a permutation  $\sigma$  on  $S$  is an Automorphism iff

$$(\forall s_1 \in S, s_2 \in S). (s_1 \rightarrow_R s_2) \Rightarrow (\sigma(s_1) \rightarrow_R \sigma(s_2))$$

If, furthermore, an Automorphism satisfies the property

$$\forall s \in S. L(s) = L(\sigma(s))$$

it is called an *Invariance Permutation*.

Auto-  
morphisms  
& Invariance  
Permuta-  
tions

Both, Automorphisms and Invariance Permutations, for a Structure  $M$  form a Group each. These Groups will be called *Automorphism Group* and *Invariance Group* for  $M$ , respectively. Actually, we still have to prove the Group Axioms (cf. 10) for these two groups, but we will postpone the proofs to exercises 13 and 14.

**Exercise 12** For a finite Set  $X$  proof that  $Sym X$  satisfies the Group Axioms from definition 10

**Exercise 13** Prove that the Automorphism Group for a Structure is closed under multiplication and inverses.

**Exercise 14** Prove the Group Axioms for Invariance Groups and their closure under multiplication and inverses.

## 4 Abstraction

Clarke, Grumberg et al considered in [2] and [1] abstraction along the most important techniques for optimizing model-checking. In this section we regard two abstraction techniques, namely *Cone-of-Influence Reduction* and *Data Abstraction*.

Both techniques exploit that states can be understood as valuations  $V \rightarrow D$ . Cone-of-Influence Reduction aims at reducing the domain of those valuations, i.e. the set of variables to those that are really needed during model-checking. Data Abstraction on the other hand leads to a reduction of the codomain  $D$ . In either case the set of all possible valuations is reduced, and thus the set of states.

## 4.1 Cone-of-Influence Reduction

In the preliminaries we mentioned that both the set of initial states  $S_0$  and the next-state relation  $R$  can be expressed in terms of first order formulae  $\mathcal{S}_0$  and  $\mathcal{R}$ . Thereby  $\mathcal{S}_0$  typically describes the values for the systems variables in order to make the corresponding state initial and can thus be seen as a (set of) instantiation(s)  $v = f(\emptyset)$ . Similarly, the formula  $\mathcal{R}$  for the next-state relation can be seen as a function associating a value to each variable  $v'$  in the successor state depending on the valuation of the variables in the predecesing state, i.e. a function  $v' = f(s)$ .

Generally, we can state that transitions and initiality can be expressed in terms of function of the following kind

$$v = f(s : V \rightarrow D)$$

It is obvious that such a function does not only describe how variables depend on the valuation of other variables, but also which variables they depend on.

We define for some variable  $v$  the variable dependency  $depend(v)$  to be the set of variables  $v'$  occuring on the righthandside of  $v = f(s)$ . Then, the transitive closure  $depend^*$  of  $depend$  is the smallest set of variables such that

- $depend^*(v) \supseteq depend(v)$
- $depend^*(v) \supseteq \bigcup \{V \mid \exists v' \in depend^*(v). V = depend(v')\}$

Based upon  $depend^*$  we can define for a set  $V$  of system variables the *Cone-of-Influence* as  $C(V) = V \cup \bigcup \{depend^*(v) \mid v \in V\}$ .

Assume for some Kripke Structure  $\langle S, R, S_0, L \rangle$  the set  $C = \{v_1, \dots, v_k\}$  is the Cone of Influence. Then the structure  $M^C = \langle S^C, R^C, S_0^C, L^C \rangle$  obtained by Cone-of-Influence Reduction from  $M$  is defined by

- $S^C = \{s^C \mid s \in S \wedge s^C = s|_C\}$  with  $s|_C$  being the restriction of  $s$  to variables in  $C$
- $s|_C \rightarrow_{R^C} t|_C$  if there are states  $s \rightarrow_R t$  with  $s^C = s|_C$  and  $t^C = t|_C$
- $L(s^C) = \{(v = d) \mid \exists s \in S. (s^C = s|_C \wedge (v = d) \in L(s) \wedge v \in C)\}$
- $S_0^C = \{s_0^C \mid s_0 \in S_0 \wedge s_0^C = s_0|_C\}$

The definitions of variable dependencies given so far are quite technical. So, let us demonstrate their meaning for our crossing.

**Example 15 (Crossing - Cone of Influence)** *Recall from example 4 that we have defined  $\mathcal{R}$  as a conjunction of several implications. An excerpt are the following formulae describing how the values of  $t_1$  and  $p_1$  are determined*

Dependencies  
between  
Variables

Reduction  
by Cone-of-  
Influence

$$\begin{aligned}
& (s_1(t_1) = \text{green} \Rightarrow (s_2(t_1) = \text{yellow} \wedge s_2(p_1) = \text{red})) \wedge \\
& (s_1(t_1) = \text{yellow} \Rightarrow (s_2(t_1) = \text{red} \wedge s_2(p_1) = \text{green})) \wedge \\
& (s_1(t_1) = \text{red} \Rightarrow (s_2(t_1) = \text{green} \wedge s_2(p_1) = \text{red})) \wedge
\end{aligned}$$

Note, that these three implications determine the function  $f$  for  $t_1$  and  $p_1$  like follows :

$$f(t_1) = \begin{cases} \text{green} : t_1 = \text{red} \\ \text{yellow} : t_1 = \text{green} \\ \text{red} : t_1 = \text{yellow} \end{cases}$$

and

$$f(p_1) = \begin{cases} \text{green} : t_1 = \text{yellow} \\ \text{red} : t_1 = \text{green} \vee t_1 = \text{red} \end{cases}$$

Note, that the right-hand sides describe the valuations in state  $s_1$ , while the left-hand sides describe the valuations in state  $s_2$ . Now, it is clear that  $t_1$  depends on itself only, and  $p_1$  depends on  $t_1$ , too. We get the following dependency relation

$$\text{depend} = \{t_i, p_i \mapsto \{t_i\} \mid i \leq 4\}$$

coinciding with its transitive closure  $\text{depend}^*$ .

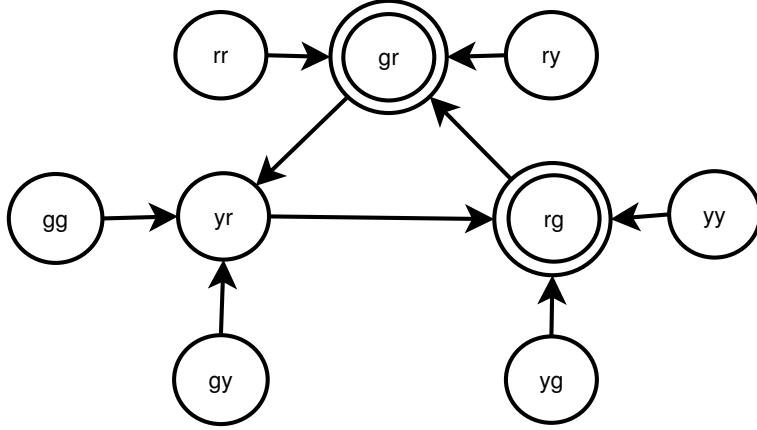
It is obviously a good idea to design a Crossing such that in no state the traffic and the pedestrians on the same street are allowed to move at the same time. This property for one street and one direction can be described by the following CTL\* formula.

$$F = AG.\neg((t_1 = \text{green}) \wedge (p_1 = \text{green}))$$

$F$  contains the variables  $V = \{t_1, p_1\}$ , such that its Cone of Influence is  $C(V) = \{t_1, p_1\}$ . Then the reduced Structure  $M^C$  is described by

$$\begin{aligned}
\bullet S &= \left\{ \begin{array}{ccc} (t_1 \mapsto \text{green}, p_1 \mapsto \text{green}) & (t_1 \mapsto \text{green}, p_1 \mapsto \text{yellow}) & (t_1 \mapsto \text{green}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{green}) & (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{yellow}) & (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{red}, p_1 \mapsto \text{green}) & (t_1 \mapsto \text{red}, p_1 \mapsto \text{yellow}) & (t_1 \mapsto \text{red}, p_1 \mapsto \text{red}) \end{array} \right\} \\
\bullet R &= \left\{ \begin{array}{ll} (t_1 \mapsto \text{green}, p_1 \mapsto \text{green}) & \rightarrow (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{green}, p_1 \mapsto \text{yellow}) & \rightarrow (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{green}, p_1 \mapsto \text{red}) & \rightarrow (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{green}) & \rightarrow (t_1 \mapsto \text{red}, p_1 \mapsto \text{green}) \\ (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{yellow}) & \rightarrow (t_1 \mapsto \text{red}, p_1 \mapsto \text{green}) \\ (t_1 \mapsto \text{yellow}, p_1 \mapsto \text{red}) & \rightarrow (t_1 \mapsto \text{red}, p_1 \mapsto \text{green}) \\ (t_1 \mapsto \text{red}, p_1 \mapsto \text{green}) & \rightarrow (t_1 \mapsto \text{green}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{red}, p_1 \mapsto \text{yellow}) & \rightarrow (t_1 \mapsto \text{green}, p_1 \mapsto \text{red}) \\ (t_1 \mapsto \text{red}, p_1 \mapsto \text{red}) & \rightarrow (t_1 \mapsto \text{green}, p_1 \mapsto \text{red}) \end{array} \right\} \\
\bullet S_0 &= \{(t_1 \mapsto \text{green}, p_1 \mapsto \text{red}), (t_1 \mapsto \text{red}, p_1 \mapsto \text{green})\} \\
\bullet L(s) &= \{(v = d) \mid v \in V \wedge s(v) = d\}
\end{aligned}$$

Or, diagrammatically (“gr” e.g. refers to state  $(t_1 \mapsto \text{green}, p_1 \mapsto \text{red})$ )



Now, we have a bisimulation equivalent Structure for  $M$  with a state space reduced from 6561 to 9 states. Of course, this example does not represent industrial cases.

In order to use Cone-of-Influence reduction in model checking we have to make sure that this kind of reduction preserves CTL\* models. This is formalized and proven in the following theorem.

**Theorem 16** *Let  $M = \langle S, R, S_0, L \rangle$  be a Kripke Structure and  $M^C = \langle S^C, R^C, S_0^C, L^C \rangle$  its Cone-of-Influence reduction. Then for any CTL\* formula  $\phi$  defined over variables in  $C$  it follows  $M \models \phi$  from  $M^C \models \phi$ .*

To prove this theorem we exploit results from theorem 8. If we can find a bisimulation  $\mathcal{B}$  with  $M \simeq_{\mathcal{B}} M^C$  then we can conclude preservation of CTL\* models as stated in the above theorem. We will use this proof technique for the later reductions as well.

We define  $\mathcal{B}$  as follows

$$(s, s^C) \in \mathcal{B} \iff s = s|_C$$

Then, to prove  $\mathcal{B}$  to be a bisimulation we have to ensure  $\mathcal{B}$  satisfies the two properties of simulation as given in definition 5

1. obviously, the labelling  $L(s^C)$  is just the restriction  $L(s)$  to propositions over variables in  $C$ , or in other words  $L(s^C) = AP^C \cap L(s)$
2. for states  $(s, s^C) \in \mathcal{B}$  with  $s \rightarrow_R t$  for some  $t \in S$  there is a  $t^C \in S^C$  with  $s^C \rightarrow_{R^C} t^C$  such that  $(t, t^C) \in \mathcal{B}$ : we obtain  $t^C = t|_C$  by definition of  $M$  and  $s^C \rightarrow_{R^C} t^C$  by the same argument.

In a second step we prove that  $\mathcal{B}^{-1}$  is also a simulation

1. focussing on variables in the cone of influence we obtain equality  $L(s) = L(s|_C)$  proving  $L(s) = AP \cap L(s|_C)$
2. for two states  $(s^C, s) \in \mathcal{B}^{-1}$  if there is a  $t^C \in S^C$  with  $s^C \rightarrow_{R^C} t^C$  then there has to be a state  $t \in S$  with  $s \rightarrow_R t$  such that  $(t^C, t) \in \mathcal{B}^{-1}$ : recall

that the description of  $R$  is restricted to variables in  $C$  only, such that for each  $s^*$  with  $(s^C, s^*) \in \mathcal{B}$  there is a bunch of  $t^*$  with  $s^* \rightarrow_R t^*$ . So we can chose for  $s$  an arbitrary  $t^*$  that reduces to  $t^C$ , i.e.  $t^C = t^*|_C$ . For such a  $t^*$  it also holds that  $(t^C, t^*) \in \mathcal{B}$  by choice.

Finally, to prove  $M \simeq_{\mathcal{B}} M^C$  we have to make sure that for each initial state  $s_0 \in S_0$  there is a corresponding initial state  $s_0^C \in S_0^C$  with  $(s_0, s_0^C) \in \mathcal{B}$  and vice versa. For the first part we can apply the construction of  $M^C$ , i.e.  $s_0^C = s_0|_C$ . For the second part we just chose one state  $s_0$  satisfying  $s_0^C = s_0|_C$ . Its existence is ensured by construction of  $M^C$ .

**Exercise 17** *Compute for our Crossing Example a Cone-of-Influence reduced Structure for a formula expressing that the traffic lights should not be set to green on both streets. Model-check this property in the reduced Structure.*

## 4.2 Data Abstraction

Often variables are defined over a domain larger than necessary. For an example assume a temperature sensor inside a chemical reactor. At values exceeding a critical limit a computer connected to such a sensor broadcasts an alarm. In this case it is not necessary to regard all the values the sensor can measure, but only whether the temperature is above or below the critical limit.

*Data Abstraction* is aimed at removing this kind of redundancies by a mapping from the actual values of the systems variables to a set of abstract data values.

### 4.2.1 Approximations

Consider a structure  $M$  with states  $s : V \rightarrow D$  describing the valuations of all system variables in  $V$  over a common domain  $D$ . Furthermore, consider a surjective abstraction function  $h : D \rightarrow A$  mapping values from  $D$  to the domain  $A$  of abstract values. For the reduced structure the states are compositions of the corresponding states  $s$  and the abstraction function  $h$ . Then they describe valuations of the form  $V \rightarrow A$ .

In dependence on the abstraction function  $h$  we define the an equivalence relation  $\sim_h$  for states  $s, t \in S$  as  $s \sim_h t \iff h \circ s = h \circ t$ , with  $\circ$  denoting the composition of functions. This equivalence relation allows us to collapse the state space into equivalence classes w.r.t.  $\sim_h$ . By applying abstraction to a Kripke Structure  $M$  one obtain the an *Approximation* of  $M$  satisfying the following definition

**Definition 18 (Approximation of a Kripke Structure)** *For an abstraction function  $h : D \rightarrow A$  the Kripke Structure  $\hat{M} = \langle \hat{S}, \hat{R}, \hat{S}_0, \hat{L} \rangle$  is an Approximation of a Kripke Structure  $M = \langle S, R, S_0, L \rangle$ , written  $M \sqsubseteq \hat{M}$  if*

- $\exists s_0 (h(s_0) = \hat{s}_0 \wedge \mathcal{S}_0(s_0))$  implies  $\hat{\mathcal{S}}_0(\hat{s}_0)$
- $\exists s_1, s_2. (\hat{s}_1 = h \circ s_1 \wedge \hat{s}_2 = h \circ s_2 \wedge \mathcal{R}(s_1, s_2)) \Rightarrow \hat{\mathcal{R}}(\hat{s}_1, \hat{s}_2)$
- for any  $\hat{s} \in \hat{S}$  there is a state  $s \in S$ , such that  $\hat{L}(\hat{S}) = \{(v = a) \mid \exists (v = d) \in L(s). a = h(d)\}$

An Approximation that contains only those initial states and transitions required by the above definition is called minimal.

**Definition 19 (Minimal Approximations)** *Let  $M = \langle S, R, S_0, L \rangle$  be a (rooted) Kripke Structure over a set  $V$  of variables over a common domain  $D$ , such that each state  $s \in S : V \rightarrow D$  is a valuation of the systems variables. Then the Kripke Structure  $M^{min} = \langle S^{min}, R^{min}, S_0^{min}, L^{min} \rangle$  is the corresponding minimal Kripke Structure if*

- $S^{min} = \{s^{min} \mid \exists (s \in S).s^{min} = (h \circ s)\}$
- $S_0^{min} = \{s_0^{min} \in S^{min} \mid \exists (s_0 \in S_0).s_0^{min} = (h \circ s_0)\}$
- $R^{min} = \{(s^{min}, t^{min}) \mid \exists (s \in S, t \in S).(s^{min} = (h \circ s) \wedge t^{min} = (h \circ t) \wedge (s \rightarrow_R t))\}$
- $L^{min}(s^{min}) = \{v^{min} = a \mid \exists (s \in S)(v = d \in L(s).s^{min} = h \circ s \wedge a = h(a))\}$

One of the most interesting features of Approximation is that for a Kripke Structure  $M$  the approximation  $\hat{M}$  simulates  $M$ . By theorem 8 we know that any ACTL\* formula  $\phi$  holding in  $\hat{M}$  holds in  $M$  as well. Formally, we state this in the following theorem

Semantical  
Properties of  
Approxima-  
tion

**Theorem 20** *For two Kripke Structures  $M$  and  $\hat{M}$ , and an abstraction function  $h : V \rightarrow A$  it follows from  $M \sqsubseteq \hat{M}$  that  $M \preceq \hat{M}$ .*

*Proof by construction of a simulation  $\mathcal{S}$  :*

We construct a simulation  $\mathcal{S} \subseteq S \times \hat{S}$  as  $(s, \hat{s}) \in \mathcal{S} \iff \hat{s} = h \circ s$ . From definition 5 we know that for  $\mathcal{S}$  to be a simulation as in  $M \preceq_{\mathcal{S}} \hat{M}$  we have to prove the two well-known properties for simulations.

However, the labelling property is not trivial to prove, since  $AP$  and  $\hat{AP}$  are in general disjoint, but the labels in  $\hat{M}$  are in general non-empty. Therefore, we will use the following trick. Assume  $f : A \rightarrow D$  to be an (injective) embedding of abstract values in  $A$  into the domain  $D$ . This embedding can be extended to an (injective) embedding  $f' : \hat{AP} \rightarrow AP$  such that  $f'(v = a) = (v = f(a))$ . Then we can construct from  $\hat{M}$  a Structure  $M'$  which is identical to  $\hat{M}$  except that  $AP' = \{(v = a') \mid \exists (v = a) \in AP.a' = f(a)\}$  and the labelling for a state  $s \in S$  the labelling in  $M'$  is given as  $L'(s) = \{p' \in AP' \mid \exists p \in L(s).p' = f'(p)\}$ . Obviously, both structures are equivalent up to renaming of values which is obviously a simulation.

Now, if we can prove  $M \preceq_{\mathcal{S}} M'$  then by transitivity of the simulation relation we know  $M \preceq_{\mathcal{S}} \hat{M}$ , too

1. From the definition of  $\hat{\cdot}$  and injection  $f$  we obtain  $L(s') = AP' \cap L(s)$
2. Let  $s \rightarrow_R t$  be a transition in  $M$ ; and there is a by definition 18 of Approximations a transition  $\hat{s} \rightarrow_{\hat{R}} \hat{t}$  with  $\hat{s} = h \circ s$  and  $\hat{t} = h \circ t$ . By construction of  $\mathcal{S}$  we know that  $\hat{s}$  is the only state in  $\hat{S}$  with  $(s, \hat{s}) \in \mathcal{S}$ , and thus we get for every state in  $\hat{S}$  with this property a successor  $t$  satisfying  $(t, \hat{t}) \in \mathcal{S}$ . By identity of the structures we can trivially complete this proof by stating that both  $\hat{s}$  and  $\hat{t}$  are uniquely determined states in  $S'$  as well, and there is a transition  $\hat{s} \rightarrow_{R'} \hat{t}$  by construction of  $M'$ .

The existential quantifications in the previous definition requires us to unwind the whole structure  $M$  to find  $M^{min}$ . In the case that  $M$  is large this task quickly becomes infeasible. In the next chapter we will see an approach to compute a good Approximation based upon the first order representations for  $S_0$  and  $R$  introduced in the preliminaries.

#### 4.2.2 Constructing Data Abstractions

Recall from the preliminaries that a Structure can be uniquely determined by two first order formulae  $\mathcal{R}$  and  $\mathcal{S}_0$ . In this sense one can construct the Approximation  $\hat{M}$  by evaluating the respective formulae  $\mathcal{R}$  and  $\mathcal{S}_0$  with respect to definitions 18 and 19.

- $\exists s_0(h(s_0) = \hat{s}_0 \wedge \mathcal{S}_0(s_0))$  implies  $\hat{\mathcal{S}}_0(\hat{s}_0)$
- $\exists s_1, s_2.(\hat{s}_1 = h \circ s_1 \wedge \hat{s}_2 = h \circ s_2 \wedge \mathcal{R}(s_1, s_2)) \Rightarrow \hat{\mathcal{R}}(\hat{s}_1, \hat{s}_2)$
- $\forall \hat{s} \in \hat{S}. \exists s \in S. \hat{s} = h \circ s \wedge \hat{L}(\hat{S}) = \{(v = a) \mid \exists (v = d) \in L(s). a = h(d)\}$

All of the three parts involve existential quantification over the state space  $S$ . However, this is computationally expensive. Clarke et al suggested in [?] to lift those existential quantifications in the first order formulae  $\mathcal{R}$  and  $\mathcal{S}_0$ . Subsequently, we define this lifting formally.

**Definition 21 (Lifting)** For  $\phi$ ,  $\phi_1$ , and  $\phi_2$  being first order formulae we define the following lifting

1. If  $P$  is a primitive proposition and  $s$  a state then  
 $\mathcal{T}(P)(\hat{s}) = \exists s \in S. (\hat{s} = h \circ s \wedge P(s))$  and  
 $\mathcal{T}(\neg P)(\hat{s}) = \neg \exists s \in S. (\hat{s} = h \circ s \wedge P(s))$
2.  $\mathcal{T}(\phi_1 \wedge \phi_2) = \mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2)$
3.  $\mathcal{T}(\phi_1 \vee \phi_2) = \mathcal{T}(\phi_1) \vee \mathcal{T}(\phi_2)$
4.  $\mathcal{T}(\forall x. \phi) = \forall \hat{x}. \mathcal{T}(\phi)$
5.  $\mathcal{T}(\exists x. \phi) = \exists \hat{x}. \mathcal{T}(\phi)$

Analogously, a Lifting can be given for binary relations.

1. If  $R$  is a primitive relation and  $s_1, s_2$  being states then  
 $\mathcal{T}(R)(\hat{s}_1, \hat{s}_2) = \exists s_1 \in S, s_2 \in S. (\hat{s}_1 = h \circ s_1 \wedge \hat{s}_2 = h \circ s_2 \wedge R(s_1, s_2))$  and  
 $\mathcal{T}(\neg R)(\hat{s}_1, \hat{s}_2) = \neg \exists s_1 \in S, s_2 \in S. (\hat{s}_1 = h \circ s_1 \wedge \hat{s}_2 = h \circ s_2 \wedge R(s_1, s_2))$
2.  $\mathcal{T}(\phi_1 \wedge \phi_2) = \mathcal{T}(\phi_1) \wedge \mathcal{T}(\phi_2)$
3.  $\mathcal{T}(\phi_1 \vee \phi_2) = \mathcal{T}(\phi_1) \vee \mathcal{T}(\phi_2)$
4.  $\mathcal{T}(\forall x. \phi) = \forall \hat{x}. \mathcal{T}(\phi)$
5.  $\mathcal{T}(\exists x. \phi) = \exists \hat{x}. \mathcal{T}(\phi)$

With  $\mathcal{T}$  we can now construct the approximating Structure  $\hat{M}$  by evaluating  $\mathcal{T}(\mathcal{S}_0)$  and  $\mathcal{T}(\mathcal{R})$  as

- $\hat{S}_0 = \{\hat{s} \in \hat{S} \mid \mathcal{T}(\mathcal{S}_0)(\hat{s})\}$
- $\hat{R} = \{(\hat{s}_1, \hat{s}_2) \in \hat{S} \times \hat{S} \mid \mathcal{T}(\mathcal{R})(\hat{s}_1, \hat{s}_2)\}$

For accurateness we prove that we did not miss details of the behaviour of  $M$  when constructing the Approximation with  $\mathcal{T}$ . Therefor we prove for any first order formula  $\phi$  that  $\phi$  implies  $\mathcal{T}(\phi)$ .

**Theorem 22** For  $M = \langle S, R, S_0, L \rangle$  a Structure and  $\hat{M} = \langle \hat{S}, \hat{R}, \hat{S}_0, \hat{L} \rangle$  its Approximation it holds that

$$\begin{aligned} \forall \hat{s} \in \hat{S}. (\exists s \in S. \hat{s} = h \circ s \wedge \mathcal{S}_0(s)) &\Rightarrow \mathcal{T}(\mathcal{S}_0)(\hat{s}) \text{ and} \\ \forall \hat{s}_1 \in \hat{S}, \hat{s}_2 \in \hat{S}. (\exists s_1 \in S, s_2 \in S. \hat{s}_1 = h \circ s_1 \wedge \hat{s}_2 = h \circ s_2 \wedge \mathcal{R}(s_1, s_2)) &\Rightarrow \mathcal{T}(\mathcal{R})(\hat{s}_1, \hat{s}_2) \end{aligned}$$

*Proof:* Can be done by structural induction over the respective formula  $\mathcal{S}_0$  or  $\mathcal{R}$ .

**Example 23 (Crossing - Approximation)** Previously, we defined a traffic light over the three states green, yellow, and red. In the real world cars shall stop when the traffic light shows red and yellow. Hence, it appears possible to reduce the state-space of a single traffic light to two states, namely stop and go.

For demonstrative purpose we focus on that substructure of our crossing structure from example 4 describing the behaviour of a single traffic light. We get the following Structure  $M_t$  over the set  $V_t = \{t\}$  of variables, and the domain  $D_t = \{\text{green, yellow, red}\}$

- $S_t = \{t \mapsto \text{green}, t \mapsto \text{yellow}, t \mapsto \text{red}\}$
- $R_t = \left\{ \begin{array}{l} (t \mapsto \text{green}) \Rightarrow (t \mapsto \text{yellow}) \\ (t \mapsto \text{yellow}) \Rightarrow (t \mapsto \text{red}) \\ (t \mapsto \text{red}) \Rightarrow (t \mapsto \text{green}) \end{array} \right\}$
- $S_{0t} = \{t \mapsto \text{green}\}$

The corresponding first order formulae are then definable as follows

- $\mathcal{S}_0(s) = (s(t) = \text{green})$
- $\mathcal{R}(s_1, s_2) = \begin{array}{l} (s_1(t) = \text{green}) \Rightarrow (s_2(t) = \text{yellow}) \wedge \\ (s_1(t) = \text{yellow}) \Rightarrow (s_2(t) = \text{red}) \wedge \\ (s_1(t) = \text{red}) \Rightarrow (s_2(t) = \text{green}) \end{array}$

Our abstraction function  $h : D \rightarrow A$  with an abstract domain  $A = \{\text{go, stop}\}$  is given by  $h(\text{green}) = \text{go}, h(\text{yellow}) = \text{stop}, h(\text{red}) = \text{stop}$ . Then the following Structure  $\hat{M}_t$  is an Approximation of  $M_t$ , and in fact also the minimal one

- $S_t = \{t \mapsto \text{stop}, t \mapsto \text{go}\}$
- $R_t = \left\{ \begin{array}{l} (t \mapsto \text{stop}) \Rightarrow (t \mapsto \text{stop}) \\ (t \mapsto \text{stop}) \Rightarrow (t \mapsto \text{go}) \\ (t \mapsto \text{go}) \Rightarrow (t \mapsto \text{stop}) \end{array} \right\}$
- $S_{0t} = \{t \mapsto \text{go}\}$

It is now an easy matter to check the properties of Approximations, such that we refer for the proof to exercise .

More interesting is how to compute the Lifting for  $\mathcal{S}_0$  and  $\mathcal{R}$ . Because  $\mathcal{S}_0$  is just an atomic proposition and thus rather trivial, we demonstrate the Lifting on  $\mathcal{R}$ .

$$\begin{aligned}
\mathcal{TR}(\hat{s}_1, \hat{s}_2) &= \\
&\mathcal{T}(\neg(s_1(t) = \text{green}) \vee (s_2(t) = \text{yellow}))(\hat{s}_1, \hat{s}_2) \wedge \\
&\mathcal{T}(\neg(s_1(t) = \text{yellow}) \vee (s_2(t) = \text{red}))(\hat{s}_1, \hat{s}_2) \wedge \quad = \\
&\mathcal{T}(\neg(s_1(t) = \text{red}) \vee (s_2(t) = \text{green}))(\hat{s}_1, \hat{s}_2) \\
&\mathcal{T}(\neg(s_1(t) = \text{green}))(\hat{s}_1, \hat{s}_2) \vee \mathcal{T}(s_2(t) = \text{yellow})(\hat{s}_1, \hat{s}_2) \wedge \\
&\mathcal{T}(\neg(s_1(t) = \text{yellow}))(\hat{s}_1, \hat{s}_2) \vee \mathcal{T}(s_2(t) = \text{red})(\hat{s}_1, \hat{s}_2) \wedge \\
&\mathcal{T}(\neg(s_1(t) = \text{red}))(\hat{s}_1, \hat{s}_2) \vee \mathcal{T}(s_2(t) = \text{green})(\hat{s}_1, \hat{s}_2)
\end{aligned}$$

Now, we want to decide whether  $(t \mapsto \text{stop}) \rightarrow_{\hat{R}} (t \mapsto \text{green})$ . Therefor we have to check if

$$\begin{aligned}
&\mathcal{T}(\neg(s_1(t) = \text{green}))((t \mapsto \text{stop}), (t \mapsto \text{green})) \vee \mathcal{T}(s_2(t) = \text{yellow})((t \mapsto \text{stop}), (t \mapsto \text{green})) \wedge \\
&\mathcal{T}(\neg(s_1(t) = \text{yellow}))((t \mapsto \text{stop}), (t \mapsto \text{green})) \vee \mathcal{T}(s_2(t) = \text{red})((t \mapsto \text{stop}), (t \mapsto \text{green})) \wedge \\
&\mathcal{T}(\neg(s_1(t) = \text{red}))((t \mapsto \text{stop}), (t \mapsto \text{green})) \vee \mathcal{T}(s_2(t) = \text{green})((t \mapsto \text{stop}), (t \mapsto \text{green}))
\end{aligned}$$

By applying the definition of  $\mathcal{T}$  for atomic propositions this is just an easy matter.

*ToDo* : expand

**Exercise 24** Prove that  $\hat{M}_t$  satisfies all properties of an Approximation.

### 4.2.3 Exact Approximations

In the latter paragraphs we have seen what Approximations are and how they can be efficiently computed by Lifting  $\mathcal{T}$  we have furthermore seen that Structures satisfy at least the same ACTL\* formulae as their respective Approximations. However, this property is rather weak when one wants to specify with CTL\* formulae. Therefor we introduce exact Approximations in the following.

**Definition 25 (Exact Approximations)** Let  $\hat{M}$  be an Approximation of  $M$  for an abstraction function  $h$ , i.e.  $M \sqsubseteq_h \hat{M}$ , then  $\hat{M}$  is the exact Approximation, written  $M \approx_h \hat{M}$ , if

1.  $\hat{s}_0 \in \hat{S}_0$  implies  $\forall s \in S. (\hat{s}_0 = h \circ s \Rightarrow s \in S_0)$
2.  $\hat{s}_1 \rightarrow_{\hat{R}} \hat{s}_2$  implies  $\forall s_1, s_2 \in S. (\hat{s}_1 = h \circ s_1 \wedge \hat{s}_2 = h \circ s_2 \Rightarrow s_1 \rightarrow s_2)$

Consequently, we have to prove that Exact Approximations preserve CTL\* models as intended. This can be done by proving  $M$  and  $\hat{M}$  bisimulation equivalent in construction a bisimulation  $\mathcal{B}$  as given by

*ToDo*

Preservation  
of CTL\*  
Models

**Example 26 (Crossing - Exact Approximation)** Consider example 23 where we have given an Approximation  $\hat{M}_t$  for the Structure  $M_t$ . Obviously,  $\hat{M}_t$  does not satisfy the second property of Exact Approximations.

As we have seen there is a transition  $(t \mapsto \text{stop}) \rightarrow_{\hat{R}_t} (t \mapsto \text{stop})$  in  $\hat{M}_t$ , and it is clear that the state  $(t \mapsto \text{red})$  in  $M_t$  composed with  $h$  results in the state  $(t \mapsto \text{stop})$  in  $M_t$ . However, there is no transition  $(t \mapsto \text{red}) \rightarrow_R (t \mapsto \text{red})$  in  $M_t$ ; contradicting the second condition for Exact Approximations.

The reader may find an abstraction function  $h$  on  $M_t$  leading to an Exact Approximation in exercise 27

**Exercise 27** Find an abstraction function  $h : D \rightarrow A$  for  $M_t$  such that the Approximation  $\hat{M}_t$  w.r.t.  $h$  is exact. Is this really worth abstracting?

## 5 Symmetry

Often complex systems exhibit great redundancy in their design. Symmetry is an attempt to filter out such redundancy and collapse the whole systems structure modulo these redundancies.

Structures are implicitly decomposed into Substructures which are compared for symmetric behaviour. Such symmetries are formalized by special bijective mappings on the state-space, namely permutations. Symmetry as an Reduction Technique is based upon a group theoretic and thus abstract treatment of those permutations

### 5.1 Theoretical Background

Assume the automorphism group  $G$  of permutations  $\sigma$  over the state space  $S$  of a Kripke Structure  $M = \langle S, R, S_0, L \rangle$ , then the  $\theta(s)$  of a state  $s \in S$  is the set of states  $t$ , such that  $\exists \sigma \in G. \sigma(s) = t$ . From each orbit  $\theta(s)$  we select a representing state  $rep(\theta(s))$ . The the quotient Structure for  $M$  is defined as follows

**Definition 28 (Quotient Models)** Let  $M = \langle S, R, S_0, L \rangle$  be a Kripke Structure, then the quotient Structure  $M^G = \langle S^G, R^G, S_0^G, L^G \rangle$  is defined as

- $S^G = \{\theta(s) \mid s \in S\}$  being the set of orbits for elements of  $S$
- $R^G = \{(\theta(s_1), \theta(s_2)) \mid (s_1, s_2) \in R\}$
- $S_0^G = \{\theta(s_0) \mid s_0 \in S_0\}$
- $L^G(\theta(s)) = L(rep(\theta(s)))$

The obvious weak point of the definition is that of the labelling function  $L^G$ . It is not independent from the choice of the representative  $rep(\theta(s))$ . Therefore we further restrict permutation group to invariance permutations as defined in the preliminary section 3.4.

In repeated manner we can prove that the original Structure satisfies at least the same CTL\* formulae as the Quotient Model does by giving a Simulation.

**Theorem 29** Let  $M = \langle S, R, S_0, L \rangle$  be a Kripke Structure and  $M^G = \langle S^G, R^G, S_0^G, L^G \rangle$  its corresponding Quotient Structure then  $M^G$  bisimulates  $M$ , i.e.  $M \simeq_h M^G$

*Proof:* by Construction of the Bisimulation

Let  $\mathcal{B} : S \times S^G$  be a binary relation being defined by  $\mathcal{B}(s, \theta(s))$  for each  $s \in S$ . Then we prove that  $\mathcal{B}$  is indeed a bisimulation. Therefor we first have to ensure the two well-known criteria from definition 5 for simulation

1.  $L(s) = L^G(\theta(s))$  :  
By definition 28 of  $M^G$  we know that  $L^G(\theta(s)) = L(rep(\theta(s)))$ . Since,  $rep(\theta(s)) \in \theta(s)$  there must be a permutation  $\sigma$ , such that  $\sigma s = (rep(\theta(s)))$ . From  $G$  being an invariance group according to definition ?? we know that for all  $p \in AP$  the equivalence  $p \in L(rep(\theta(s))) \iff p \in L(s)$ .
2. from  $(s, s^G) \in \mathcal{B}$  and  $s \rightarrow_R t$  then there is a  $t^G \in S^G$  with  $s^G \rightarrow_{RG} t^G$  such that  $(t, t^G) \in \mathcal{B}$  :  
By construction of  $B$  we know  $s^G = \theta(s)$  and by definition 28 we know that for any  $s \rightarrow_R t$  there is a pair  $\theta(s) \rightarrow_{RG} \theta(t)$ . By construction of  $\mathcal{B}$  we know that  $(t, \theta(t)) \in \mathcal{B}$ .

In a second step we have show that  $\mathcal{B}^{-1}$  is a simulation as well.

1.  $L^G(\theta(s)) = L(s)$  follows from prior argument
2. from  $(\theta(s), s) \in H^{-1}$  and  $\theta(s) \rightarrow_{RG} \vartheta$  it follows the existance of a  $t' \in S$  such that  $s \rightarrow_R t'$  and  $(\vartheta, t') \in H^{-1}$  :  
By knowing that orbits are non-empty we know that  $\vartheta$  has at least one state, namely  $rep(\vartheta)$ . In the following let  $t = rep(\vartheta)$ , such that  $\vartheta$  can be expressed by  $\theta(t)$  which gives us  $\theta(s) \rightarrow_{RG} \theta(t)$ . By definition 28 we conclude there are  $s^*, t^* \in S$  such that  $s^* \in \theta(s)$ ,  $t^* \in \theta(t)$ , and  $s^* \rightarrow_R t^*$ . Because  $s^* \in \theta(s)$  there must be a permutation  $\sigma$  with  $\sigma s^* = s$ . By definition ?? we get  $\sigma s^* \rightarrow_R \sigma t^*$  or equivalently  $s \rightarrow_R \sigma t^*$ . Notice that  $\sigma t^*$  and  $t$  belong to the same orbit such that  $\sigma t^*$  is the state  $t'$  we are looking for.

## 5.2 Model Checking with Symmetry

In this section we describe an approach to employ the ideas of symmetry in model checking. Obviously, one of the most vital parts of such an algorithm is the efficient computation of the orbit of a state.

Let  $M = \langle S, R, S_0, L \rangle$  be a Kripke Structure over variables  $V$  and  $G$  be the automorphism group  $G$  on  $M$  with  $r$  generators  $\sigma_1, \sigma_2, \dots, \sigma_r$  being automorphisms on  $S$ . Then the orbit relation  $\Theta(x, y) \iff (x \in \theta(y))$  can be computed as the least fixed point of the following equation

$$Y(x, y) = (x = y \vee \exists z. (Y(x, z) \wedge \bigvee_i y = \sigma_i(z)))$$

That the generators  $\sigma_1, \dots, \sigma_r$  are indeed automorphisms can be determined according to definition 11 by checking if  $R(s, t)$  is equivalent to  $R(s \circ \sigma_i, t \circ \sigma_i)$ . Remember from the preliminaries that  $R$  can be represented by a first order formula, which allows the feasible decision of this question.

Once we have a representation of the set  $S^G$  of orbits for permutations over  $S$  we can find a choice function  $f : S \rightarrow S$  mapping a state  $s$  to a representative state of  $s$ ' orbit. Thereby the state space can be collapsed to orbits.

**Example 30 (Crossing - Symmetry)** *Obviously, our Crossing Example contains a lot of Symmetry. As an example we choose the following permutation*

$$\begin{aligned} \sigma(s_1) &= s_2 \wedge \sigma(s_2) = s_1 \text{ with} \\ s_1(t_1) &= s_1(t_3) = s_2(t_2) = s_2(t_4) \wedge \\ s_2(t_1) &= s_2(t_3) = s_1(t_2) = s_2(t_4) \wedge \\ s_1(p_1) &= s_1(p_3) = s_2(p_2) = s_2(p_4) \wedge \\ s_2(p_1) &= s_2(p_3) = s_1(p_2) = s_2(p_4) \wedge \end{aligned}$$

$\sigma(s) = s$  for all other states  $s$ .

$\sigma$  proves to be an automorphism, too, as it can be shown by applying definition 11.

The permutation Group generated by  $\{\sigma\}$  does only contain  $\sigma$  and the identity permutation  $id$ .

Exemplifying the Orbit Problem we want to show that

- $s_1 = \{t_1, t_3, p_2, p_4 \mapsto \text{green}; p_1, p_3, t_2, t_4 \mapsto \text{red}\}$  and
- $s_2 = \{t_2, t_4, p_1, p_3 \mapsto \text{green}; t_1, t_3, p_2, p_4 \mapsto \text{red}\}$

are in the same orbit. It is just a matter of instantiation to show that  $s_2 = \sigma(s_1)$  and vice versa, such that we obtain  $\Theta(s_1, s_2)$  as the Least Fixed Point of the recursively defined relation  $Y(s_1, s_2)$  in the first recursion level.

**Exercise 31** *Are there any other Automorphisms for the Crossing Structure?*

### 5.3 Complexity Issues

We conclude this section on Symmetry with some remarks on complexity issues in the application of Symmetry. We first roughly sketch a proof that the orbit problem is at least as hard as the graph isomorphism problem known to be in NP.

Let  $G$  be a permutation group over the set  $\{v_1, \dots, v_n\}$ , and let  $G$  be represented by a finite set of generators. The *Orbit Problem* consists in deciding for two vectors  $x, y \in B^n$  over the base  $B$  whether there is a permutation  $\sigma \in |G|$  such that  $y = \sigma(x)$ .

For deciding the complexity of the Orbit Problem one constructs a polynomial reduction to the Graph Isomorphism Problem.

Given two graphs  $\Gamma_1 = (V_1, E_1)$  and  $\Gamma_2 = (V_2, E_2)$  such that  $V_1$  and  $V_2$  are of the same cardinality, the Graph Isomorphism Problem consists in computing a graph isomorphism, i.e. a bijection  $f : V_1 \rightarrow V_2$  that satisfies the following structural property

$$(i, j) \in E_1 \iff (f(i), f(j)) \in E_2$$

The Orbit Problem

The Graph Isomorphism Problem

## 6 Conclusions

As we have seen in this report Abstraction and Symmetry are strong tools in optimizing a Structure for Model Checking. However, apart from Cone-of-Influence Reduction all techniques relied on a given heuristics. Data Abstraction requires a good Abstraction Function, and Symmetry requires a good Set of Generators for the Invariance Group not to run into an everlasting Model Checking Procedure.

## References

- [1] Edmund M. Clarke, Jr. Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2001.
- [2] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.