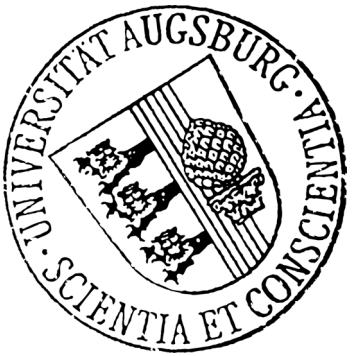
Bisimulations and Model Refinement

Dissertation zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat)

der Fakultät für Angewandte Informatik
der Universität Augsburg



Roland Glück
University of Augsburg

Erstgutachter: Prof. Dr. Bernhard Möller
Zweitgtuachter: Prof. Dr. Robert Lorenz

In Memory of my Father

Contents

Acknowledgement

First I want to thank all people who supported and motivated me during the work on this thesis.

The first person I am grateful to is my supervisor Prof. Dr. Bernhard Möller, Universität Augsburg, who supported my academic career and guided me through my scientific work. He also enabled my work by the position at the Universität Augsburg I had six years.

Another person I want to mention is Michel Sintzoff who passed away in 2011. He put my attention to the topic of this thesis and was till his premature death always open for discussions both face to face and via email.

I am grateful to my colleagues Han-Hing Dang, Martin Endres, Patrick Roocks, Florian Wenzel and Andreas Zelend for valuable and fruitful discussions and the good mood at our chair. Beside this, I am especially grateful to Dominik Köppl for thorough proofreading of this thesis.

A great motivation were the meetings of the community of the RelMiCS and RAMiCS conferences where I had the opportunity to present and discuss the recent status of my work. I am also grateful to the anonymous referees of these conferences who pointed out some mistakes and overlooked some others but always accepted my submissions.

Last but not least I am grateful to my parents and my family who supported me not only financially but also morally during the last years.

Chapter 1

Introduction

This chapter gives a rough idea of the approach which we will use in this book. Our main goal is to construct a system with a desired property by removing 'bad' transitions from a given system. However, the given system may be a large, even infinite one. The idea is to reduce the state number of the given transition system by constructing a system with equivalent (with respect to the property under consideration) dynamic behaviour. Then the problem is solved for the reduced system, and subsequently a solution for the initial system is constructed by means of the solution for the reduced system. We give an abstract algorithm, which will be instantiated and slightly adapted for some problem classes we investigate in the following chapters.

1.1 Motivation and General Idea

The problem of refining a transition system by removing undesirable transitions is among others a central topic in control theory (see e.g. [Ber00, Son90, Vin00] for general ideas; we will give more special references in the respective chapters). The field of problems studied there ranges from general dynamic systems over hybrid systems to automata and Petri nets. In the case of dynamic and hybrid systems one often has to deal with a continuous component, usually modeled by differential equations. We will

focus here on systems which are describable without such a continuous component. Especially, we concentrate on systems with a huge or even infinite number of states as they appear in internet routing, certain game graphs or specifications for the temporal behaviour of large systems. The goal of the refinement procedure is to obtain a subsystem with a desired optimality, safety, temporal or other comparable property. In contrast to model checking we do not focus on the *verification* of given system properties but on the *construction* of system properties.

An example for this kind of problem is finding an optimal (in most cases shortest) way between two locations on the surface of the earth, using streets, bridges, ships and so on, as offered by Google. In this case the application of the classical algorithm by Dijkstra fails to the mere size of the input. So other approaches were developed, as for example in [DGW13] and [DW13]. However, these approaches are tailored for online-applications and take advantage of intensive precomputation. We will present another idea which tries to solve a given problem on a smaller system and out of it constructs a solution for the original problem.

A natural wish is to reduce the state number of a transition system in a way that does not change its behaviour with respect to a desired property without giving away too much information. In this case we can hope that an algorithm computing a suitable subsystem has a better runtime if we apply it to the reduced system. Of course we have to ensure that we can construct a subsystem of the original system in an adequate way. This leads to a *shrink-refine-expand*-approach: first, we reduce the given systems size by a suitable construction. Second, we run a known refinement algorithm on the reduced system to ensure the given property on this system. Third, we expand the output of the previous step into a subsystem of the given system by a suitable refinement operation. In an abstract way, this approach can be concentrated in Algorithm 1.

Algorithm 1 Abstract Generic Algorithm

Require: A Refinement Algorithm A for ensuring property P is known, M is a Huge Transition System and B is a Bisimulation Equivalence for M

1: $M/B \leftarrow$ shrinking of M by means of B
2: $(M/B)' \leftarrow A(M/B)$ // apply A to M/B, resulting in $(M/B)'$
3: $M' \leftarrow$ expansion of $(M/B)'$ by means of B

Ensure: M' fulfills property P

In this abstract algorithm we have to find concrete methods for the shrinking (Line 1) and expansion (Line 3). As possible choice we will introduce in Chapter 6 the *bisimulation quotient* for the shrinking and the *expansion* for the expanding operation. The use of bisimulation quotients for verification of system properties has a long tradition

for example in model refinement (see [BK08]) or automata theory (see [Myh57, Ner58]) and serves there as standard tool. As we assume that algorithms for the refinement step are available (nevertheless we will discuss also this step for some cases; this will be useful for discussing the complexity of the approach) we have to specify also a bisimulation based expanding operation. This operation is obtained from the quotient operation by means of a Galois connection.

To obtain a correct version of our shrink-refine-expand-approach we can not use an arbitrary bisimulation but have to find a bisimulation that is compatible in some sense with the desired property. Trivially, the identity is always a correct choice, since it is a bisimulation and does not change the system at all, but it does not help in decreasing the magnitude of the system. The key property is that compatible bisimulations are closed under composition, relational converse and union, so there is always a coarsest compatible bisimulation, which is also an equivalence. The number of states will be reduced by building the quotient depending on how coarse the bisimulation is. So a coarser bisimulation (which merges more states into one state of its quotient than a finer one) will reduce the state number more than a finer one. A better runtime can be expected if the constructing of the quotient is faster than the immediate execution of the refinement algorithm (as we will see, the expansion can be done in linear time). For transition systems M with n states and m transitions we will use an algorithm which computes B and hence M/B in $\mathcal{O}(m \cdot log(n))$. So Algorithm 1 will lead to a runtime improvement only if A has a runtime in $\Omega(m \cdot log(n))$. Whether a speed up can be achieved depends heavily on the degree of compression by constructing M/B. Of course, there is never a guarantee that the quotient has a smaller state number than the original problem. This depends on the structure of the given system; a reduction can be expected especially for well-structured, hierarchical or tree-like systems.

The shrink-refine-expand-approach can also be useful if the system under consideration has an infinite number of states (compare here the work presented e.g. in [BBR04, HHW95, JKM98]). In this case a refinement algorithm with finite runtime is in general not available (if one is interested in the problem of shortest paths then the Dijkstra algorithm is not applicable to an infinite system). If building the quotient yields a system with a finite state number the problem is transformed into a solvable one. Of course, there is never a guarantee that the quotient has a smaller state number than the original problem. This depends on the structure of the given system; a reduction can be expected especially for well-structured, hierarchical or tree-like systems.

Clearly, the correctness of Algorithm 1 depends on the choice of the bisimulation equivalence B and the property P. In this work we will identify some problem classes for which the algorithm is correct (if a suitable bisimulation equivalence is chosen). Unfortunately, we will not present a general criterion whether a problem can be tackled by this approach or not.

1.2 Organisation

To give the reader a first guide through this thesis we give an overview of its organisation.

In **Chapter 2** we introduce and clarify our writings in order to avoid misunderstandings. The following **Chapter 3** is dedicated to the exemplary construction of a control policy in a simple infinite transition system and motivates the use of bisimulation equivalences. The control policy is obtained by use of a bisimulation equivalence as in Algorithm 1 but without computing the bisimulation equivalence explicitly.

The formal definition of bisimulations takes place in **Chapter 4**. Some namings and definitions are also given in this chapter. A formal framework for transition systems is introduced in **Chapter 5**. Similarly to the previous chapter it contains mainly definitions and writing conventions.

The terms from Chapter 4 and Chapter 5 are put together in **Chapter 6**. There also the formal definition of the shrinking and expansion operation takes place but still in an abstract generic way without concerning concrete refinement goals.

A first class of concrete problems is considered in **Chapter 7**. It deals with properties similar to terms known from temporal logic. The desired goal is to ensure reachability of or abidance in a set of 'good' states.

An important and comprehensive class of problems is the topic of **Chapter 8**. Here we show that a large class of optimality problems, including shortest path and bottleneck problems, are compatible with our approach. The following **Chapter 9** gives in some points a generalisation of Chapter 8; whereas in Chapter 8 the argumentation is done in a pointwise manner Chapter 9 uses an approach via a kind of adjacency matrices and employs their algebraic properties.

Chapter 10 deals with so called stochastic games. This problem class is known to be in $\mathcal{NP} \cap \mathsf{co}\mathcal{NP}$ but there is no provably polynomial algorithm known; so even shrinking by one node can lead to a speed up.

An algebraic formulation of our approach is given in **Chapter 11**. There we use semirings to model relations and partitions and investigate the relationship between this abstract structure and concrete relations and sets. Some of the ideas from Chapter 11 appear again in **Chapter 12** where we investigate how the algebraic characterisations from Chapter 11 can be used as input for automated prove systems.

Chapter 13 gives a summary of this thesis and discusses future work and open questions. Finally, in **Appendix A** we give a tabular survey of the results we obtained in the experiments concerning automated theorem proving as described in Chapter 12.

Chapter 2

Writing Conventions

In this chapter we introduce some writing conventions and namings we use throughout this book. They deviate in some cases from the usual ones, so we give them to avoid misunderstandings and ease the reading. In reasonable cases we will deviate slightly from these conventions; the highest goals are both exactness and readability of notation.

Throughout this work we use capital letters (D, T, V ...) for sets. The elements of a set are denoted by lower case letters (v, x, y, ...), if necessary equipped with indices. A set of the form $\{i \in \mathbb{N} \mid m \leq i \leq n\}$ is abbreviated as $\{m..n\}$. Scripted letters ($\mathcal{V}$) denote families of sets (but we will not denote every system of sets by scripted letters). For a set V a family of sets $\mathcal{V} = \{V_i \mid i \in I\}$ is called a *partition* of V if $\bigcup\limits_{i \in I} V_i = V$ and $i = j \Leftrightarrow V_i \cap V_j \neq \emptyset$ for all $i, j \in I$ hold. To avoid notational clumsiness we use the abbreviation $\bigcup \mathcal{V}$ for $\bigcup\limits_{V \in \mathcal{V}} V$.

Functions are denoted by lower case letters (a, f, g, ...). For a function $f : M \to N$ and a subset $M' \subseteq M$ of M we denote by $f|_{M'}$ the *restriction* of f to M', defined by $f|_{M'} : M' \to N$ with $f|_{M'}(m) = f(m)$ for all $m \in M$. Given a function $f : (M \times N) \to O$ we use the notation $f(m, n)$ instead of the correct $f((m, n))$.

Relations, i.e. subsets of a cartesian product $M \times N$, are also sets and hence denoted by capital letters, too (mostly R, S, but also by others). The converse of a relation R is denoted by R° (i.e. $R^\circ = \{(y, x) \mid (x, y) \in R\}$) and the composition of two relations

R and S by $R;S$. We also write xRy instead of $(x,y) \in R$. The identity relation over a set M is denoted by id_M, i.e. $\mathsf{id}_M = \{(m,m) \mid m \in M\}$. A relation $R \subseteq M \times N$ is called *left-total* if $I_M \subseteq R;R^\circ$, and *right-total* if $I_N \subseteq R^\circ;R$ holds. For a relation R and an element x we denote the *image* of x under R by $xR = \{y \mid xRy\}$. The *preimage* of x under R is defined by $Rx = xR^\circ$. By pointwise lifting to sets we define the image and preimage of a set N under R by $NR = \bigcup_{x \in N} xR$ and $RN = \bigcup_{x \in N} Rx$.

A *graph* G is a tuple $G = (V,E)$ where V is an arbitrary set of *node*, and $E \subseteq V \times V$ is the set of *edges*. So the term graph denotes explicitly a directed graph. A graph is called *finite* if its node set is finite. Often we distinguish graphs by indices ($G_1 = (V_1,E_1)$, $G_2 = (V_2,E_2)$, ...) and different nodes from the same graph by superscripted indices, so different nodes in V_1 are written v_1^1, v_1^2, and so on. This convention holds also for other sequences, in particular for the symbols of a word. We will use the terms image and preimage also for graphs in a canonical manner.

A *walk* w in a graph $G = (V,E)$ is a nonempty sequence of nodes $v^1v^2 \dots v^n$ such that $(v^i, v^{i+1}) \in E$ holds for all $i \in \{1..n-1\}$. We will also consider infinite walks which are infinite sequences of nodes $v^1v^2v^3 \dots$ with $(v^i, v^{i+1}) \in E$ for all $i \in \mathbb{N}^+$.

The *edge length* of a walk $v^1v^2 \dots v^n$ is simply $n-1$. A *path* is a walk $v^1v^2 \dots v^n$ with $i = j \Leftrightarrow v_i = v_j$ for all $i,j \in \{1..n-1\}$. Finally, a *cycle* is a walk $v^1v^2 \dots v^n$ with edge length at least one and $v^i = v^j \Leftrightarrow i = j \vee |j-i| = n-1$ for all $i,j \in \{1..n\}$. Given two walks $w_1 = w_1^1w_1^2 \dots w_1^n$ and $w_2 = w_2^1w_2^2 \dots w_2^m$ we define the *concatenation* or *gluing* $w_1 \bowtie w_2$ by $w_1 \bowtie w_2 = w_1^1w_1^2 \dots w_1^{n-1}w_2^1w_2^2 \dots w_2^m$ if $w_1^n = w_2^1$ and declare it as undefined if $w_1^n \neq w_2^1$. We say that a node y is *reachable* from a node x if there exists a walk $v^1v^2 \dots v^n$ with $x = v^1$ and $y = v^n$. Note that this predicate is not symmetric. The set of all walks in G between x and y is denoted by $W_G(x,y)$, and the set of all paths by $P_G(x,y)$. If G is clear from the context it will also be omitted. A graph $G' = (V',E')$ is called a *subgraph* of a graph $G = (V,E)$, denoted by $G' \preccurlyeq G$, if $V' \subseteq V \wedge E' \subseteq E$ holds.

For relations and graphs also consult the encyclopedic work [SS93].

As usual, the end of a proof is marked by a black square (■). Sometimes we want to insert remarks into the text. The end of such a remark is indicated by a square (□)

Chapter 3

Example

As a first example we solve a simple single-player game with an infinite number of states. The solution takes place via an ad hoc approach without theoretical background. The observations of this chapter will serve as a motivating base for the more abstract terms in the further course.

We consider a simple single-player game, played on the interval $[3, 10] \subseteq \mathbb{R}$. A number in $[3, 8[$ can be increased by one, and additionally a number in $[4, 5]$ can also be doubled. The goal is to obtain a number in the target interval $[8, 10]$. However, increasing a number by one causes a cost of one unit, and doubling a number costs two units. Of course, we want to reach the target interval with the lowest possible costs.

Our task is to construct a strategy for this game, i.e., a policy that for every number in $[3, 10]$ indicates whether it should be increased by one or doubled. A classical approach like the Dijsktra or Floyd-Warshall algorithm will not work for now, since the state space is infinite. So we start with a backward analysis of the game. First, the target interval can be reached in one step via two possibilities: it can be reached from any number in $[7, 8[$ by increasing it by one, and from any number in $[4, 5]$ by doubling. This preliminary observation can be illustrated as in Figure 3.1.

Doing the next step back, we can reach any number in $[4, 5]$ by adding one to a number in $[3, 4]$, and any number in $[7, 8[$ by increasing a number in $[6, 7[$ by one. This yields the situation as in Figure 3.2. However, the number 4 is contained both in the node

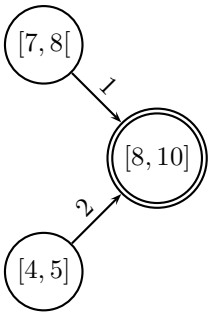


Figure 3.1: First Stage

associated with the interval $[4, 5]$ and the one associated with $[3, 4]$. So the number 4 and all its predecessors and successors considered till now have to be treated in a special way. Therefore we remove 3 from the interval $[3, 4]$, 4 from both $[3, 4]$ and $[4, 5]$, and 5 from $[4, 5]$ and obtain the new intermediate result shown in Figure 3.3.

Because the preimage of $[6, 7[$ under the game's rules is $[5, 6[$ we obtain an intermediate result, which can be seen in Figure 3.4 (the other nodes have only empty preimages). Here we have the same problems with the number 5 as before with 4: it is contained both in $[5, 6[$ and $\{5\}$. An analogous procedure leads to the situation in Figure 3.5.

Now we are almost done. In our backward analysis we have to determine the preimage of the interval $]5, 6[$. Fortunately, this is the interval $]4, 5[$ which is already contained in our previous intermediate result. So after inserting the edge between these two nodes we obtain the final result from Figure 3.6.

Note that the number 3 has a behaviour which is unique among all others under consideration: the target interval can be reached by increasing it twice by one and subsequent doubling.

We reached our goal and reduced the infinite game to a finite one, so we can apply an arbitrary shortest path algorithm on the finite game, which leads to the policy shown in Figure 3.7 (black arrows indicate the optimal strategy). If we transform this result back to the original game we obtain as the optimal strategy doubling each number in $[4, 5]$ and increasing every other number by one.

Let us take a closer look to our approach. We write $x \xrightarrow{+1}_g y$ and $x \xrightarrow{\cdot 2}_g y$ resp. if x

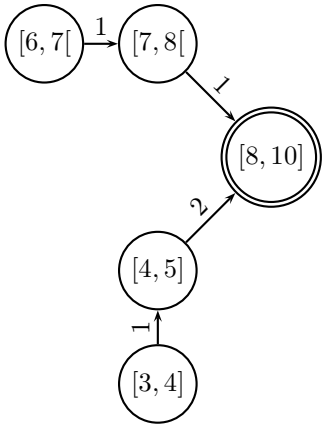


Figure 3.2: Second Stage

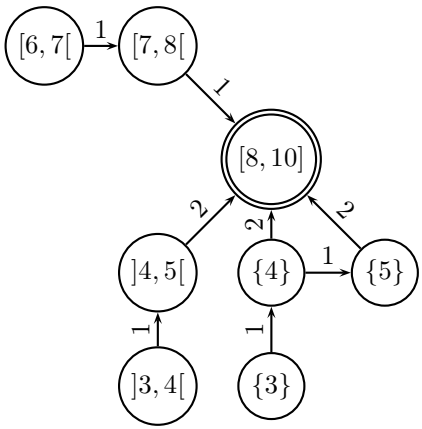


Figure 3.3: Situation after Treating 4

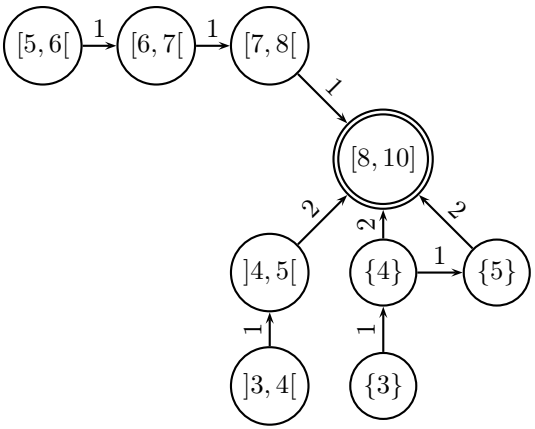


Figure 3.4: Third Stage

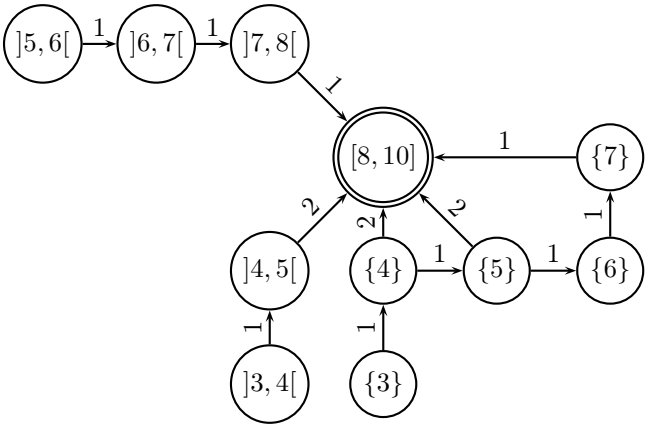


Figure 3.5: Situation after Treating 5

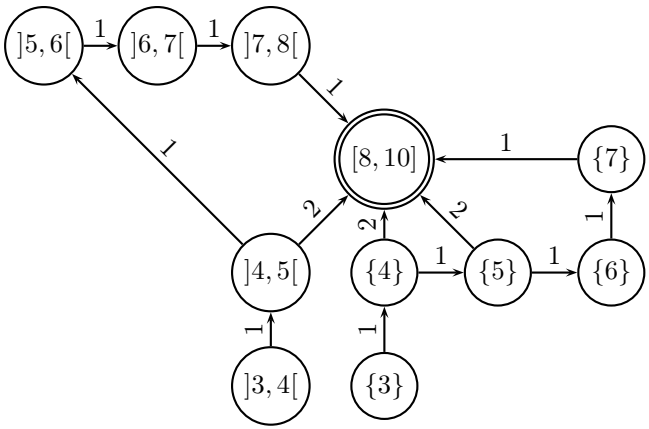


Figure 3.6: Final Stage

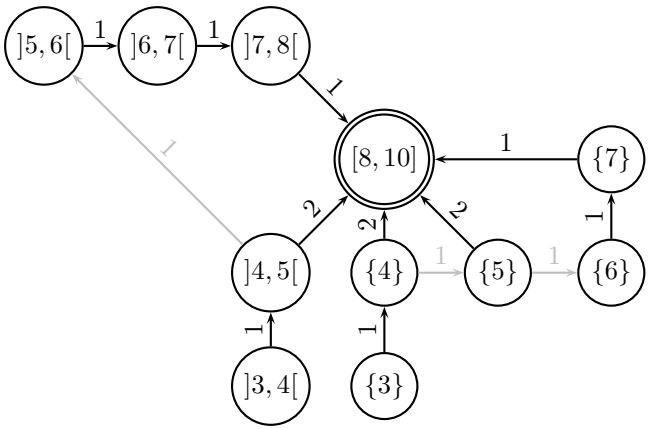


Figure 3.7: Strategy

can be transformed into y by increasing it by one resp. doubling it according to the rules. For two sets X and Y from the nodes' captions of Figure 3.6 we write $X \xrightarrow{+1}_b Y$ and $X \xrightarrow{\cdot 2}_b Y$ with analogous meanings (so we have for example $\{4\} \xrightarrow{\cdot 2}_g [8, 10]$ and $]5, 6[\xrightarrow{+1}_b]6, 7[$). Then the following properties hold:

- $x \xrightarrow{+1}_g y \wedge x \in X \Rightarrow \exists Y : y \in Y \wedge X \xrightarrow{+1}_b Y$
- $x \xrightarrow{\cdot 2}_g y \wedge x \in X \Rightarrow \exists Y : y \in Y \wedge X \xrightarrow{\cdot 2}_b Y$
- $X \xrightarrow{+1}_b Y \wedge X \ni x \Rightarrow \exists y : Y \ni y \wedge x \xrightarrow{+1}_g y$
- $X \xrightarrow{\cdot 2}_b Y \wedge X \ni x \Rightarrow \exists y : Y \ni y \wedge x \xrightarrow{\cdot 2}_g y$

This means that the original game and the game from Figure 3.6 have an equivalent behaviour in a certain sense. We will state this more precisely in the next chapter.

Chapter 4

Set-labelled Graphs and Bisimulations

In this chapter we introduce two basic concepts: Set-labelled graphs and bisimulations between set-labelled graphs. Set-labelled graphs correspond to labelled transition systems without node labels. For later running time considerations we present a data structure for set-labelled graphs which allows the execution of basic operations in optimal time. Dynamically equivalent behaviour of two set-labelled graphs can be witnessed by a bisimulation between them. A special case of bisimulations are autobisimulations, i.e., bisimulations between a set-labelled graph and itself. They can be used to identify nodes with equivalent dynamic behaviour.

4.1 Set-labelled Graphs

4.1.1 Basic Definitions

A basic ingredient of our game from Chapter 3 was a labelled graph. In the further course it turns we will see that one label per edge (as in our introductory example)

can be insufficient, so we will allow multiple labels on an edge. This is captured by the following definition:

Definition 4.1.1 Let $G = (V, E)$ be a graph, M be an arbitrary nonempty set and $g : E \to \mathcal{P}(L)\backslash\emptyset$ a mapping such that $\bigcup_{(v,w)\in E} g(v, w) = L$. Then the pair (G, g) is called a *set-labelled graph*. g is called its *labelling function* and L its *label set*.

For an edge $(x, y) \in E$ and a label $\ell \in L$ we may also write $x \xrightarrow{\ell}_g y$ if $\ell \in g(x, y)$ holds. This writing corresponds to the usual notion for labelled transition systems. In the context of set-labelled graphs L will always denote its label set. A labelled graph is called *uniquely labelled* if $|g(v, w)| = 1$ holds for all edges (v, w). This corresponds to conventional labelled graphs. A *finitely labelled graph* is a set-labelled graph $((V, E), g)$ with the property $|g(v, w)| < \infty$ for all $(v, w) \in E$ (note that this does not imply $|L| < \infty$ but the other way round).

Sometimes we want to restrict the edge set of a set labelled graph to edges with a certain set of labels. So for a set labelled graph $G = ((V, E), g)$ and an $\ell \in L$ we introduce the *ℓ-restricted graph* $G_\ell = ((V_\ell, E_\ell), g_\ell)$ by $V_\ell = V$, $E_\ell = \{(u, v) \in E \,|\, g(u, v) = \ell\}$ and $g_\ell(u, v) = \ell$ for all $(u, v) \in E_\ell$. For a node $v \in V$ and an $\ell \in L$ we define the set $\delta(v, \ell)$ of *ℓ-successors* by $\delta(v, \ell) =_{df} \{w \,|\, v \xrightarrow{\ell}_g w\}$. The set $g_{out}(v)$ of *out-labels* of a node v is defined by $g_{out}(v) =_{df} \{\ell \,|\, \delta(v, \ell) \neq \emptyset\}$.

Contrary to traditional labelled graphs we do not have a unique labelling along a walk in G. Therefore we have to deal with a set of labellings of a walk. So given a labelled graph (G, g) and a walk $w = w^1 w^2 \dots w^n$ in G we define the *set of labellings* $L(w) \subseteq L^*$ of w in G by $\ell^1 \ell^2 \dots \ell^{n-1} \in L(w) \Leftrightarrow \forall i \in \{1 \dots n-1\} : \ell^i \in g(w^i, w^{i+1})$. In the case of a uniquely labelled graph the set of labellings of a walk contains exactly one element. Here we use the notation $l(w)$ to denote the single element of $L(w)$.

4.1.2 A Data Structure for Set-labelled Graphs

From an algorithmic point of view it is indispensable to have a suitable data structure for set-labelled graphs at our disposal which enables us to carry out the needed operations asymptotically as fast as possible. Since the presented algorithms are run only on finite set-labelled graphs (except possibly the construction of the quotient of an infinite system as described in Definition 6.2.1, which is done by symbolic execution as e.g. in [BBR04, HHW95, JKM98]) we restrict our discussion to finite set-labelled graphs. In detail, we have the following requirements:

1. For a node $v \in V$ and an edge label $\ell \in L$, determine $\delta(v, \ell)$ in $\Theta(|\delta(v, \ell)|)$ time.

2. For a pair of nodes (v, w), determine $g(v, w)$ in $\Theta(|g(v, w)|)$ time.

3. For an edge $(v, w) \in E$ and an edge label $\ell \in L$, remove the edge label ℓ from $g(v, w)$ in $\mathcal{O}(1)$ time. If $g(v, w) = \{\ell\}$ then (v, w) is also removed from E. In the case $\ell \notin g(v, w)$ the data structure remains unchanged.

4. Remove an edge (v, w) (together with its labels) from E in $\Theta(|g(v, w)|)$ time. If $(v, w) \notin E$ holds the data structure remains unchanged.

5. Test the existence of an edge in $\mathcal{O}(1)$ time.

6. For a pair of nodes (v, w) and an edge label $\ell \in L$, add ℓ to $g(v, w)$ in $\mathcal{O}(1)$ time. If $(v, w) \notin E$ holds then (v, w) is added to E afore. In the case $\ell \in g(v, w)$ no changes are performed.

7. For a node $v \in V$, determine vE in $\mathcal{O}(|vE|)$ time.

Despite of Operation 4 the required running times are the best possible ones. Even in the case of removing an edge together with its labels, the required running time seems rather natural.

The Operations 1 - 5 and Operation 7 can be used for queries in algorithms and refinement operations (cf. Definition 5.2.1), whereas Operation 6 is suitable for building up a data structure representing a set-labelled graph by successive adding of labelled edges.

There are tools such as `mCRL2` (see [CGK+13, GMvWU06, mCL]) or `LTSA` (details can be found in [UCKM03, FUMK06, LTS]) which support some of the above-specified operations. However, their main focus lies on verification of a system, so they do not support explicitly Operations 3 and 4. Moreover, there are no remarks about the running time of the supported operations. We will now construct a data structure which fulfills all the above requirements.

Adjacency matrices support testing the existence of an edge in constant time (Operation 5) and removing an edge (Operation 4) and can be modified to support Operation 2 in the desired time. The other operations take longer time if we operate on adjacency matrices. On the contrary, adjacency lists have drawbacks at these operations but guarantee the other operations in the desired time. The proposed data structure which we call a *mixed representation* combines both adjacency matrices and adjacency lists. This will now be described in detail.

For the construction we assume w.l.o.g. that $V = \{0 \dots n-1\}$ and $L = \{0 \dots o-1\}$ hold. Then the data structure consists of the following parts:

1. An $n \times o$-array Δ of doubly linked lists of integers.

2. An $n \times n$-array Γ of doubly linked lists of integers.
3. An $n \times n \times o$-array Λ whose entries are triples of a boolean value, a pointer to an element of a list in Δ and a pointer to an element of a list in Γ.
4. An n-array Σ of doubly linked lists of integers.
5. An $n \times n$-array Φ of tuples of integers and pointers to elements of a list in Σ.

The array Δ contains at position (v, ℓ) the set $\delta(v, \ell)$ as a duplicate-free doubly linked list, so Operation 1 can be executed in the required time. Analogously, Γ contains at position (v, w) a duplicate-free doubly linked list representing $g(v, w)$, so Operation 2 can also be done in the required time. If $v \xrightarrow{\ell}_g w$ holds then the boolean value of $\Lambda[v][w][\ell]$ is true, and the two pointers point to the respective list elements of $\Delta[v][\ell]$ and $\Gamma[v][w]$. Otherwise (i.e. if $v \xrightarrow{\ell}_g w$ does not hold) the Boolean value of $\Lambda[v][w][\ell]$ is false, and both pointers are null. The doubly linked list $\Sigma[v]$ contains all successors of v (the update operations will be described later). So Operation 7 can be done with the required running time. In the integer part $\Phi[v][w]$ we find $|g(v, w)|$. It equals 0 if $(v, w) \notin E$, so testing whether $(v, w) \in E$ (Operation 5) reduces to the test $\Phi[v][w] \neq 0$ which can be done in constant time. The pointer at $\Phi[v][w]$ points to the list element with value w of the list $\Sigma[v]$ if $\Phi[v][w] > 0$ (i.e., if $(v, w) \in E$), and equals null otherwise.

The remaining Operations 3, 4 and 6 are a little bit more elaborate:

> Operation 3: First we look up the boolean value of $\Lambda[v][w][\ell]$ in $\mathcal{O}(1)$ time. If this value equals false we are done. Otherwise we execute the following operations:
>
> 1. Decrement the integer part of $\Phi[v][w]$ by 1.
> 2. If the integer part of $\Phi[v][w]$ equals 0 then w is removed from $\Sigma[v]$ and the pointer of $\Phi[v][w]$ is set to null.
> 3. Delete from the list $\Delta[v][\ell]$ the element with value w.
> 4. Delete from the list $\Gamma[v][w]$ the element with value ℓ.
> 5. Set the boolean part of $\Lambda[v][w][\ell]$ to false and its two pointers to null.
>
> Note that the deletion operations can be executed in constant time because we have pointers to the respective elements of the doubly linked lists.
>
> Operation 4: We set the integer part of $\Phi[v][w]$ to 0, remove w from $\Sigma[v]$ and set the pointer of $\Phi[v][w]$ to null. Then we iterate through the list $\Gamma[v][w]$ and execute for every value ℓ in this list Steps 3 - 5 of Operation 3 as described above. Clearly, the required running time is kept.

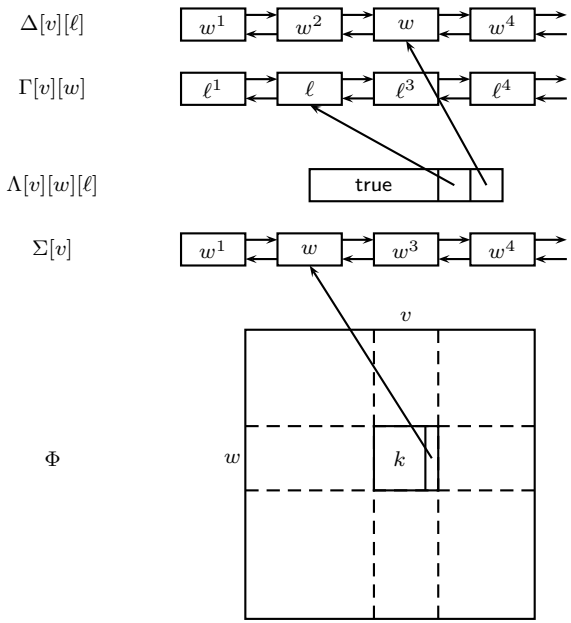


Figure 4.1: Part of a Mixed Representation before Removing an Edge Label

Operation 6: First we look up the boolean value of $\Lambda[v][w][\ell]$ in $\mathcal{O}(1)$ time. If this value equals true no further operations are necessary. Otherwise we proceed as follows:

1. Increment $\Phi[v][w]$ by 1.
2. If $\Phi[v][w]$ equals 1 we add an element with value w to $\Sigma[v]$ and set the pointer of $\Phi[v][w]$ to this element.
3. Set the boolean part of $\Lambda[v][w][\ell]$ to true.
4. Add a new element with value w to $\Delta[v][\ell]$ and let the Δ-pointer of $\Lambda[v][w][\ell]$ point to this element.
5. Add a new element with value ℓ to $\Gamma[v][w]$ and let the Γ-pointer of $\Lambda[v][w][\ell]$ point to this element.

Obviously, all these operations can be executed in constant time.

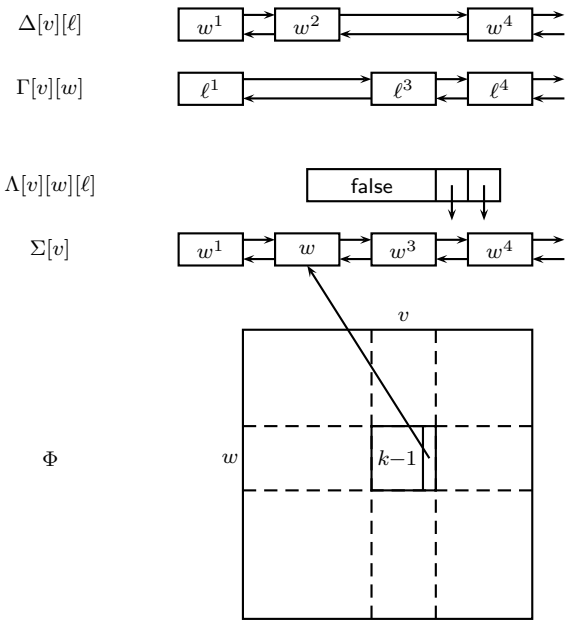


Figure 4.2: Part of a Mixed Representation after Removing an Edge Label

We will demonstrate the execution of Operation 3 (removing a label from an edge) graphically. Figure 4.1 shows part of a mixed representation, corresponding to the representation of the transition $v \xrightarrow{\ell}_g w$. The topmost list is the list $\Delta[v][\ell]$ whereas the second list corresponds to $\Gamma[v][w]$. In the middle we see the structure $\Lambda[v][w][\ell]$ with the boolean value true and pointers to the respective elements of $\Delta[v][\ell]$ and $\Gamma[v][w]$. The subjacent list is $\Sigma[v]$. On the bottom we find the matrix Φ with the integer entry k at position $\Phi[v][w]$ and a pointer to the element w in the list $\Sigma[v]$. After removing label ℓ from the edge (v, w) we obtain the situation depicted in Figure 4.2. From list $\Delta[v][\ell]$ the element with value w is removed, and from list $\Gamma[v][w]$ the element with value ℓ. In $\Lambda[v][w][\ell]$ the boolean value is set to false, and both pointers become null. Finally, the value $\Phi[v][w]$ is decremented by one to $k-1$. Here we assumed that $k > 1$ holds, otherwise we have to remove the element with value w from $\Sigma[v]$ and to set the pointer at $\Phi[v][w]$ to null. Analogously one can visualise Operation 6, the addition of a label to an edge.

The main drawback of this data structure is its space requirement of $\Theta(n^2 o)$. This is an illustration of the classical space-time tradeoff, similar to the advantages and disadvantages of adjacency matrices and adjacency lists.

4.2 Bisimulations for Set-labelled Graphs

We will now introduce one of the main concepts of this thesis. The term 'bisimulation' has become general knowledge and it is hard to name its inventor. In the bibliography of the DBLP (see [DBL]) the oldest listed item with bisimulation in its title is dated from 1985 ([Cas85]). But there are several predecessors which use the same or a very similar construction, sometimes under others names like 'behavioural equivalence relation', 'observational equivalence' or 'strong congruence' (see [BR83, KS90, Mil80, MM79, Plo76]) in the context of process algebra or 'automaton transformation' ([Myh57, Ner58]) in the context of automata theory.

Definition 4.2.1 Let $G_1 = ((V_1, E_1), g_1)$ and $G_2 = ((V_2, E_2), g_2)$ be two set-labelled graphs with the same label set L. A both left- and right-total relation $B \subseteq V_1 \times V_2$ is called a *bisimulation* between G_1 and G_2 if the following properties hold for all $\ell \in L$:

1. $\forall x_1, y_1, x_2 : x_1 \xrightarrow{\ell}_{g_1} y_1 \wedge x_1 B x_2 \Rightarrow \exists y_2 : x_2 \xrightarrow{\ell}_{g_2} y_2 \wedge y_1 B y_2$
2. $\forall x_2, y_2, x_1 : x_2 \xrightarrow{\ell}_{g_2} y_2 \wedge x_2 B^\circ x_1 \Rightarrow \exists y_1 : x_1 \xrightarrow{\ell}_{g_1} y_1 \wedge y_2 B^\circ y_1$

So the relation $\in$ is a bisimulation between the original game and the game from Depiction 3.6 from the previous section.

An relation algebraic equivalent definition is that $B^\circ; \xrightarrow{\ell}_{g_1} \subseteq \xrightarrow{\ell}_{g_2}; B^\circ$ and $B; \xrightarrow{\ell}_{g_2} \subseteq \xrightarrow{\ell}_{g_1}; B$ hold for all $\ell \in L$. We will revisit this in Chapter 11 if we give an algebraic approach to bisimulations.

Two set-labelled graphs G_1 and G_2 are called *bisimilar* if there is a bisimulation between G_1 and G_2. Because we want to investigate equivalent states of a system we are interested in bisimulations between a set-labelled graph and itself. Such a bisimulations is called an *autobisimulation*. An autobisimulation which is also an equivalence relation is called a *bisimulation equivalence*.

Often the nodes of a set labelled graph belong to different classes, so in our example we had numbers in the target interval and numbers outside of it. It this case it seems reasonable to consider bisimulation equivalences which relate only nodes of the same classes. So let $((V, E), g)$ be a set labelled graph and $\mathcal{V} = (V_i)_{i \in I}$ be a partition of its node set. We say that an autobisimulation B *respects* $\mathcal{V}$ if for all $v_1, v_2 \in V$ the

implication $v_1 B v_2 \Rightarrow \exists i \in I : v_1 \in V_i \wedge v_2 \in V_i$ holds. An equivalent formulation is that every V_i is the union of suitable equivalence classes of B.

The set of bisimulations respecting a fixed partition is closed under union and taking the converse. Moreover, the identity relation is an autobisimulation respecting every partition. So there is a greatest (with respect to the subset order) bisimulation equivalence respecting a given partition, namely the union of all bisimulation equivalences which respect this partition. Often we will need as a subroutine in our algorithms the computation of this bisimulation equivalence or of its equivalence classes. For this task, there is an efficient algorithm, developed by Paige and Tarjan (see [PT] and the work relying hereon in [Fer90]) which computes the equivalence classes of the coarsest bisimulation equivalence in $\mathcal{O}(|E| \cdot log(|V|) \cdot |L|)$ time.

One key point of the algorithms from [PT, Fer90] and related algorithms is the fact that the equivalence classes of a bisimulation form a stable partition. Formally this means the following:

Definition 4.2.2 Let $G = ((V, E), g)$ be a set labelled graph. A partition $\mathcal{V} = \{V_i \mid i \in I\}$ of V is called *stable* with respect to G if for every $\ell \in L$ and every $V_i \in \mathcal{V}$ the set of ℓ-predecessors of V_i, i.e. the set $\{u \in V \mid uE_\ell \cap V_i \neq \emptyset\}$, can be written as the union $\bigcup_{j \in J} V_j$ for a suitable subset $J \subseteq I$.

The algorithms for computing the coarsest bisimulation for a set labelled graph respecting a given partition proceed by refining the initially given partition until a stable partition is obtained. The crucial point is the following lemma:

Lemma 4.2.1 *Let $G = ((V, E), g)$ be a set labelled graph and B a bisimulation equivalence for G. Then V/B forms a stable partition of V with respect to G.*

In particular, this means that for every $(v/B) \in V/B$ every node $v' \in (v/B)$ has the same predecessors under every edge label. The algorithms from [PT, Fer90] are refinements of a partitioning algorithm given e.g. in [AHU74] and are based on clever maintaining the sets in intermediate results. The reader may also compare the construction of the quotient in our introductory example with the algorithm from [AHU74] and assure himself that the obtained partition is indeed a stable one (the initial partition was $\{[3, 8[, [8, 10]\}$, and the finally obtained one $\{\{3\},]3, 4[, \{4\},]4, 5[, \{5\},]5, 6[, \{6\},]6, 7[, \{7\},]7, 8[, [8, 10]\}$).

Chapter 5

Models

This short chapter introduces the idea of a model, a generalisation of the structure we used in our example game. We motivate the definition and introduce some basic notions and operations we will use in the sequel. Among them we introduce the important idea of model refinement.

5.1 Models

If we take a look at our example from Chapter 3 we observe that set-labelled graphs are not a suitable tool for its description. The problem is that this concept lacks the possibility to define the target set of our example, or more generally, all nodes are treated as equal. This motivates the following definition of a model:

Definition 5.1.1 A *model* M is a pair $M = (G, a)$ where $G = ((V, E), g)$ is a set-labelled graph and $a : V \to A$ is the *node labelling function* of M. G is called the *associated* set-labelled graph of M.

The node labelling function can be used for different purposes. So it can be used to model start- and/or finite states of a system as in finite automata, or it can describe a certain kind of import at a node. Also Kripke structures can be seen as models whose node labelling functions describe mappings from the node set into a set of atomic

propositions, cf. e.g. [CGP01]. The combination of two or more such meanings can be achieved by using suitable tuples as image of a. Of course, if one is not really interested in the labelling of the node or edge set one will choose a dummy function and ignore it at later stages of reasoning about the model. We will do so in Chapter 10 where we reason about stochastic games. In cases where one wants to describe costs in a network or labelled graph the associated set-labelled graph of the model under consideration is often a uniquely labelled one. So a model with a uniquely labelled associated graph is also called *uniquely labelled*.

In some cases we will use for A the power set $\mathcal{P}(X)$ of a certain set X. Then every $x \in X$ induces a subset $V_x \subseteq V$ of the node set, given by $V_x =_{df} \{v \in V \mid x \in a(v)\}$. Here we often will use only x instead of V_x, especially if x is denoted by an uppercase letter (cf. Chapter 7).

Analogously to graphs we define isomorphism between models. So two models $M_1 = (((V_1, E_1), g_1), a_1)$ and $M_2 = (((V_2, E_2), g_2), a_2)$ are *isomorphic* if there is a bijection $f : V_1 \to V_2$ with the following properties:

1. $\forall v_1, w_1 \in V_1 : (v_1, w_1) \in E_1 \Leftrightarrow (f(v_1), f(w_1)) \in E_2$
2. $\forall (v_1, w_1) \in E_1 : g_1(v_1, w_1) = g_2(f(v_1), f_(w_1))$
3. $\forall v_1 \in V_1 : a_1(v_1) = a_2(f(v_1))$

Point 1 stipulates that the graphs of the two models are isomorphic. Point 2 and 3 require that images under f of edges and nodes, resp., bear the same label. Note that due to bijectivity of f the above conditions are equivalent to the following ones:

1. $\forall v_2, w_2 \in V_2 : (v_2, w_2) \in E_2 \Leftrightarrow (f^\circ(v_2), f^\circ(w_2)) \in E_1$
2. $\forall (v_2, w_2) \in E_2 : g_2(v_2, w_2) = g_1(f^\circ(v_2), f^\circ(w_2))$
3. $\forall v_2 \in V_2 : a_2(v_2) = a_1(f^\circ(v_2))$

5.2 Model Refinement and Submodels

In our introducing example we had the task to construct a policy for a given system by removing undesired transitions. As an additional possibility we might remove some labels from a transition. This procedure is captured in the context of models in the following definition:

Definition 5.2.1 Given two models $M_1 = (((V_1, E_1), g_1), a_1)$ and $M_2 = (((V_2, E_2), g_2), a_2)$ we say that M_2 is a *submodel* of M_1 or that M_2 *refines* M_1 (also written $M_2 \preceq M_1$) if the following conditions are fulfilled:

1. $V_2 = V_1$
2. $a_2 = a_1$
3. $E_2 \subseteq E_1$
4. $g_2(v, w) \subseteq g_1|_{E_2}(v, w)$

The set of all submodels of a model M is denoted by $Sub(M)$. It is straightforward to see that $\preceq$ is a partial order on $Sub(M)$ and that $(Sub(M), \preceq)$ forms a complete lattice.

Chapter 6

Models and Bisimulations

In this chapter we investigate the interplay between bisimulations and models. First, we will extend the idea of bisimulations from set labelled graphs to models. Subsequently, we introduce autobisimulations, i.e., bisimulations between a model and itself, and investigate how a model with equivalent dynamic behaviour, the so-called quotient, can be constructed using an autobisimulation equivalence. As a kind of inverse of this operation we define the expansion operation. Finally, we introduce a generic method for obtaining a refinement of a model using a refinement of a quotient of it.

6.1 Bisimulations for Models

If we take a look back at our introductory example we see that no number of the target interval was related to a non-target interval of the reduced system, and vice versa, so in an abstract way only nodes with the same label are related to each other. This additional constraint to general bisimulations for set labelled graphs gives rise to the following definition:

Definition 6.1.1 Let $M_1 = (((V_1, E_1), g_1), a_1)$ and $M_2 = (((V_2, E_2), g_2), a_2)$ be two models. A left- and right-total relation $B \subseteq V_1 \times V_2$ is a bisimulation between M_1 and M_2 if the following holds:

1. B is a bisimulation between $((V_1, E_1), g_1)$ and $((V_2, E_2), g_2)$
2. $\forall v_1 \in V_1, v_2 \in V_2 : v_1 B v_2 \Rightarrow a_1(v_1) = a_2(v_2)$

The purpose of Requirement 1 is obvious. Item 2 stipulates that only nodes with the same node label values are related to each other. Analogously to the case of bisimulations we say that two models M_1 and M_2 are *bisimilar* if a bisimulation between them exists. As a symbol for the bisimilarity relation between models we use the sign $\sim$. Clearly, $\sim$ is an equivalence relation. This follows easily from the properties of bisimulations. We also write $M_1 \sim_B M_2$ if B is a bisimulation between M_1 and M_2 and call B a *bisimulation witness* for M_1 and M_2.

Bisimilar models have the same dynamic behaviour with respect to their edge and node labels. This is formally stated in the next lemma:

Lemma 6.1.1 *Let $M_1 = (((V_1, E_1), g_1), a_1)$ and $M_2 = (((V_2, E_2), g_2), a_2)$ be two models with $M_1 \sim_B M_2$. Let $w_1 = v_1^1 v_1^2 v_1^3 \dots v_1^n$ be a walk in (V_1, E_1) and consider an arbitrary $l_1 \in L_1(w_1)$. Then for every v_2^1 with $v_1^1 B v_2^1$ there is a walk $w_2 = v_2^1 v_2^2 v_2^3 \dots v_2^n$ in (V_2, E_2) with the following properties:*

1. *$l_1 \in L_2(w_2)$*
2. *$a_1(v_1^m) = a_2(v_2^m)$ for all $m \in \{1 \dots n\}$ and*
3. *$v_1^m B v_2^m$ for all $m \in \{1 \dots n\}$.*

Proof: Let M_1, M_2 and B be as above, and let $l_1 = \ell_1^1 \ell_1^2 \dots \ell_1^{n-1}$. Then it suffices for Part 1 to show $\ell_1^i \in g_2(v_2^i, v_2^{i+1})$ for all $i \in \{1 \dots n-1\}$. This can be shown together with the other two claims by simple induction over n.

For $n = 1$ let $v_1^1 v_2^2$ be a walk in (V_1, E_1) with $\ell_1^1 \in L_1(v_1^1 v_1^2)$ and $v_2^1 \in V_2$ a node with $v_1^1 B v_2^1$. Since B is a bisimulation there is a node v_2^2 in V_2 such that $v_1^2 B v_2^2$ and $v_2^1 \xrightarrow{\ell_1^1}_{g_2} v_2^2$ hold. Due to Requirement 2 of Definition 6.1.1 we have also $a_1(v_1^1) = a_2(v_2^1)$ and $a_1(v_1^2) = a_2(v_2^2)$.

Consider now a walk $w_1 = v_1^1 v_1^2 v_1^3 \dots v_1^n v_1^{n+1}$ in M_1 and an $l_1 = \ell_1^1 \ell_1^2 \dots \ell_1^n \in L_1(w_1)$. Due to the induction hypothesis there is a walk $w_2 = v_2^1 v_2^2 v_2^3 \dots v_2^n$ in M_2 with $\ell_1^1 \ell_1^2 \dots \ell_1^{n-1} \in L_2(w_2)$ for all $m \in \{1 \dots n-1\}$ and $a_1(v_1^m) = a_2(v_2^m)$ for all $m \in \{1 \dots n\}$; so it remains to show that there is a node v_2^{n+1} with $v_2^n \xrightarrow{\ell_1^n}_{g_2} v_2^{n+1}$ and $a_1(v_1^{n+1}) = a_2(v_2^{n+1})$. The existence of a node with the first property follows from the left-totality of a bisimulation and Part 1 of Definition 4.2.1. The second property is a consequence of Part 2 of Definition 6.1.1. ■

Remark: This Lemma fixes an error in Lemma 3.3. in [Glü11]. There, in an analogous context, the equality $g_1(v_1^i, v_1^{i+1}) = g_2(v_2^i, v_2^{i+1})$ is claimed. A counterexample is provided by the two bisimilar models in Figure 6.2: the edge $(\{a,b\},\{a,b\})$ in the right model is labelled by $\{1,2\}$ whereas there is no edge at all with this label in the left model. □

An immediate consequence is the following corollary which follows from the fact that $\sim$ is an equivalence relation and hence in particular symmetric:

Corollary 6.1.1 *Let $M_1 = (((V_1, E_1), g_1), a_1)$ and $M_2 = (((V_2, E_2), g_2), a_2)$ be two models with $M_1 \sim_B M_2$ and consider two nodes $v_1^1 \in V_1$ and $v_2^1 \in V_2$ with $v_1^1 B v_2^1$. Then a walk $v_1^1 v_1^2 v_1^3 \dots v_1^n$ in M_1 with edge label sequence $\ell^1\ell^2\dots\ell^{n-1}$ and node label sequence $a^1a^2a^3\dots a^n$ exists iff there is a walk $v_2^1 v_2^2 v_2^3 \dots v_2^n$ in M_2 with edge label sequence $\ell^1\ell^2\dots\ell^{n-1}$ and node label sequence $a^1a^2a^3\dots a^n$.*

6.2 Autobisimulations and Quotient Operation

A bisimulation between a model M and itself is called an *autobisimulation* for M. An autobisimulation on a model M which is also an equivalence relation is called a *bisimulation equivalence* for M.

A bisimulation equivalence for a model $(((V, E), g), a)$ is a bisimulation equivalence for the set labelled graph $((V, E), g)$ which respects the partition of V induced by the node labelling function a. This follows immediately from Part 2 of Definition 6.1.1.

Bisimulation equivalences on a model relate in a certain sense nodes with equivalent behaviour. This means that the equivalence class of a node can serve as a representative for the behaviour of all its nodes. This idea was already exploited in the seminal papers [Myh57] and [Ner58] and is also a welcome tool in model checking, cf. e.g. [BK08]. The formal idea behind this is the quotient construction:

Definition 6.2.1 Let $M = (((V, E), g), a)$ be a model and B a bisimulation equivalence for M. The *quotient* M/B of M by B is the model $M/B = (((V/B, E/B), g/B), a/B)$, defined as follows:

1. $(v/B, w/B) \in E/B \Leftrightarrow \exists v' \in v/B, w' \in w/B : (v', w') \in E$
2. $\ell \in (g/B)(v/B, w/B) \Leftrightarrow \exists v' \in v/B, w' \in w/B : \ell \in g(v', w')$
3. $a/B(v/B) = a(v)$

Note that a/B is well defined due to the fact that B relates only nodes with the same value of a, as stated in Requirements 2 of Definition 6.1.1. In our example a quotient is shown in Figure 3.6.

It is a well known fact that a labelled transition system and an arbitrary one of its quotients are bisimilar. This carries over to models in an easy way; for the sake completeness we give it here as a separate lemma together with its proof.

Lemma 6.2.1 *Let $M = (((V, E), g), a)$ be a model and B a bisimulation equivalence for M. Then M and M/B are bisimilar, and $\in$ is a bisimulation between M and M/B.*

Proof: First we show that $\in$ is a bisimulation between $((V, E), g)$ and $((V/B, E/B), g/B)$ (i.e. Part 1 of Definition 6.1.1). Therefore let $x, y \in V$, $X \in V/B$ and $\ell \in L$ be arbitrary with $x \stackrel{\ell}{\to}_g y$ and $x \in X$. We claim that $Y := y/B$ fulfills $X \stackrel{\ell}{\to}_{g/B} Y$ and $y \in Y$. First, $y \in Y$ is trivial. For $X \stackrel{\ell}{\to}_{g/B} Y$ we have to show that $(X, Y) \in E/B$ and $\ell \in (g/B)(X, Y)$ hold. The first property holds due to part 1 of Definition 6.2.1, and the second property because of part 2 of the same definition.

Conversely, assume $X, Y \in V/B$, $x \in V$ and $\ell \in L/B$ with $X \stackrel{\ell}{\to}_{g/B} Y$ and $X \ni x$. Then we have to show the existence of a y with $Y \ni y$ and $x \stackrel{\ell}{\to}_g y$. The Parts 1 and 2 of Definition 6.2.1 show the existence of an $x' \in X$ and $y' \in Y$ with $x' \stackrel{\ell}{\to}_g y'$. By definition of the quotient we have $x'Bx$, and since B is a bisimulation this implies the existence of a y with $x \stackrel{\ell}{\to}_g y$ and $y'By$, which in particular means $Y \ni y$. So $\in$ is a bisimulation between $((V, E), g)$ and $((V/B, E/B), g/B)$.

For the rest of Definition 6.1.1 assume $x \in V$ and $X \in V/B$ with $x \in X$. By Part 3 of Definition 6.2.1 we have $(a/B)(X) = a(x)$, so Part 2 of Definition 6.1.1 is satisfied. ■

An important property is the compatibility of bisimulation equivalence and the preimage operator on equivalence classes. This means informally that the preimage operator distributes over the union of equivalence classes. A formal statement about this is the next theorem:

Theorem 6.2.1 *Let $G = (((V, E), g, a))$ be a model and B a bisimulation equivalence for G. Consider now two arbitrary subsets $(V/B)' \subseteq (V/B)$ and $(V/B)'' \subseteq (V/B)$ of V/B and an arbitrary $\ell \in L$. Then the following equalities hold:*

1. $E_\ell(\bigcup(V/B)') = \bigcup(E/B)_\ell(V/B)'$
2. $E_\ell((\bigcup(V/B)') \cap (\bigcup(V/B)'')) = \bigcup(E/B)_\ell((V/B)' \cap (V/B)'')$

Remark: Remember the writing conventions for the preimage and the union of set families from Chapter 2. □

Before we prove this theorem we will visualise its claim. In the upper left part of Figure 6.1 the partition induced by a bisimulation equivalence is symbolised by dashed lines. The light grey area in this part indicates the union of some of these equivalence classes (corresponding to the expression $\bigcup(V/B)'$ in the first item of Theorem 6.2.1). Now we can determine the preimage of the light grey area in two ways: First, we can forget about the equivalence classes and compute the preimage directly by evaluating the term $E_\ell(\bigcup(V/B)')$ (this is shown in the lower left part; the thin arrows stand for the relevant part of the relation E_ℓ). Second, we can compute the preimages of all equivalence classes in $(V/B)'$ under $(E/B)_\ell$, e.g., evaluating the expression $\bigcup(E/B)_\ell(V/B)'$ (as in the upper right part; $(E/B)_\ell$ is represented by thick arrows). Clearly, these two possibilities lead to the same result (the lower right part). The second item of Theorem 6.2.1 has an analogous meaning.

Proof: We begin with the first equality.
'$\subseteq$': Consider an arbitrary fixed $u \in E_\ell(\bigcup(V/B)')$. Then there is a $(v'/B) \in (V/B)'$ and a $v \in (v'/B)$ such that $u \xrightarrow{\ell}_E v$ holds. Because of $u \in (u/B)$ (remember that B is an equivalence) and $v \in (v'/B)$ we have $(u/B) \xrightarrow{\ell}_{E/B} (v'/B)$. So (u/B) is an element of the preimage of (v'/B) under $(E/B)_\ell$ and hence contained in $\bigcup(E/B)_\ell(V/B)'$.
'$\supseteq$': Let $u \in \bigcup(E/B)_\ell(V/B)'$ be arbitrarily chosen. Then there is a (u'/B) with $u \in (u'/B)$ and a $(v'/B) \in (V/B)'$ with $(u'/B) \xrightarrow{\ell}_{(E/B)} (v'/B)$. Because B is an equivalence we have even $(u'/B) = (u/B)$. Due to Lemma 6.2.1 there is a $v \in (v'/B)$ such that $u \xrightarrow{\ell}_E v$ holds. Herefrom the claim follows easily.
Let us now turn to the second equality. First we take a look at the left side and argue as follows:

$$
\begin{aligned}
& u \in E_\ell((\textstyle\bigcup(V/B)') \cap (\bigcup(V/B)'')) \Leftrightarrow \\
& \quad \{ \text{ definition of the preimage } \} \\
& \exists v \in ((\textstyle\bigcup(V/B)') \cap (\bigcup(V/B)'')) : u \xrightarrow{\ell}_E v \Leftrightarrow \\
& \quad \{ \text{ set theory } \} \\
& \exists v' : (v'/B) \in (V/B)' \wedge (v'/B) \in (V/B)'' \wedge v \in (v'/B) \wedge u \xrightarrow{\ell}_E v \Leftrightarrow \\
& \quad \{ B \text{ is an equivalence } \} \\
& \exists v : (v/B) \in (V/B)' \wedge (v/B) \in (V/B)'' \wedge u \xrightarrow{\ell}_E v \qquad (1)
\end{aligned}
$$

For the right side we do the following argumentation:

$$
\begin{aligned}
& u \in \textstyle\bigcup(E/B)_\ell((V/B)' \cap (V/B)'') \Leftrightarrow \\
& \quad \{ \text{ set theory } \} \\
& \exists (u'/B) \in \textstyle\bigcup(E/B)_\ell((V/B)' \cap (V/B)'') : u \in (u'/B) \Leftrightarrow \\
& \quad \{ B \text{ is an equivalence } \} \\
& (u/B) \in (E/B)_\ell((V/B)' \cap (V/B)'') \Leftrightarrow \\
& \quad \{ \text{ definition of the preimage } \}
\end{aligned}
$$

$$\exists(v/B) : (v/B) \in (V/B)' \wedge (v/B) \in (V/B)'' \wedge (u/B) \xrightarrow{\ell}_{E/B} (v/B) \Leftrightarrow$$
$$\quad \{ \text{ definition of the quotient } \}$$
$$\exists v : (v/B) \in (V/B)' \wedge (v/B) \in (V/B)'' \wedge (u/B) \xrightarrow{\ell}_{E/B} (v/B) \qquad (2)$$

So it remains to show the equivalence of (1) and (2) (this is not totally trivial since the v's are bounded by quantifiers and need not be the same). Clearly, (1) implies (2) due to the definition of the quotient (choose for the v from (2) the same v as in (1)). From (2), we know by Lemma 6.2.1 the existence of a $v' \in (v/B)$ with $u \xrightarrow{\ell}_E v'$. This v' can be chosen for the v from (1). ■

Remark: Clearly, Part 1 of Theorem 6.2.1 follows from Part 2 of the same theorem if we choose two identical subsets of V/B. We gave a separate proof which enlightens the connections between a labelled graph and its node on the one side and one of its quotients and the equivalence classes on the other side (the proof of Part 2 remains very formal). □

A technically annoying aspect of the quotient is that the quotient of a uniquely labelled model need not be a uniquely labelled again. As an example we consider the model $M = (((V, E), g), a)$ with $V = \{a, b, c\}$, $E = \{(a, a), (a, b), (b, a), (b, b), (a, c), (b, c)\}$, $g(a, a) = g(b, b) = \{1\}$, $g(a, b) = g(b, a) = \{2\}$, $g(a, c) = g(b, c) = \{3\}$, $a(a) = a(b) =$ tr and $a(c) = \mathsf{fin}$. This model is depicted in the left part of Figure 6.2. As drawing convention, nodes with the node label fin are doubly surrounded, those with node label tr are simply surrounded.

However, the relation $B = \{(a, a), (a, b), (b, a), (b, b), (c, c)\}$ is a bisimulation equivalence for M. The resulting quotient M/B is depicted in the right part of Figure 6.2. But this is not a uniquely labelled model, since the loop $(\{a, b\}, \{a, b\})$ carries the label $\{1, 2\}$. So the construction in our introductory example was guided by a little bit of luck since we did not run into this problem.

As shown above, the label sets of an edge (v, w) and the edge $(v/B, w/B)$ in a quotient are in general not equal, but at least there is a subset relation between them as stated in the following lemma:

Lemma 6.2.2 *Let $M = (((V, E), g, a))$ be a model and B a bisimulation equivalence for M. Then for every $(v, w) \in E$ the set inclusion $g(v, w) \subseteq (g/B)(v/B, w/B)$ holds.*

Proof: Let $(v, w) \in E$ be arbitrarily chosen and fix an arbitrary $\ell \in g(v, w)$. Then we have $v \in v/B$ and $w \in w/B$ because B is an equivalence relation. By Part 1 of Definition 6.2.1 we have $(v/B, w/B) \in E/B$. Again due to $v \in v/B$ and $w \in w/B$ and Part 2 of Definition 6.2.1 we obtain $\ell \in (g/B)(v/B, w/B)$. ■

The first part of this proof shows also the following corollary:

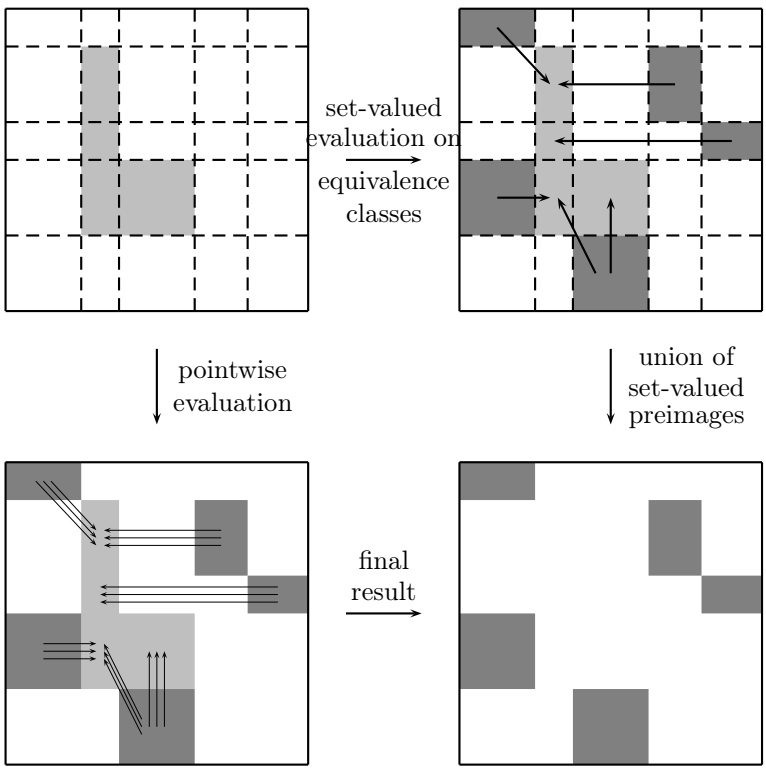


Figure 6.1: Two ways for computing set valued preimages

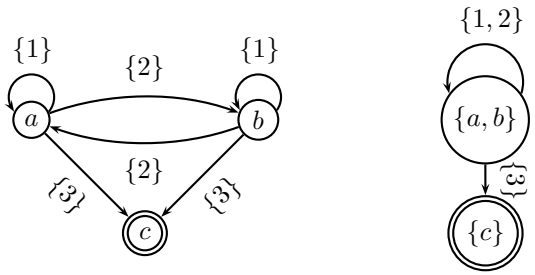


Figure 6.2: A uniquely labelled model (left) and one of its quotients (right)

Corollary 6.2.1 *Let $M = (((V, E), g, a))$ be a model and B a bisimulation equivalence for M. Then for all $v, w \in V$ the implication $(v, w) \in E \Rightarrow (v/B, w/B) \in E/B$ holds.*

6.3 Expansion Operation

The next step in our example was to generate a policy (which means a submodel) for the original system from a policy for the shrunken one. This will be done by the expansion operation:

Definition 6.3.1 Let $M = (((V, E), g), a)$ be a model, B a bisimulation equivalence for M and $(M/B)' = ((((V/B)', (E/B)'), (g/B)'), (a/B)')$ a submodel of M/B. Then we define the *expansion* $(M/B)'\backslash B = (((V', E'), g'), a')$ of $(M/B)'$ by B as follows:

1. $V' = V$
2. $(v, w) \in E' \Leftrightarrow (v, w) \in E \wedge (v/B, w/B) \in (E/B)' \wedge \exists \ell : \ell \in (g/B)'(v/B, w/B) \wedge \ell \in g(v, w)$
3. $\ell \in g'(v, w) \Leftrightarrow \ell \in g(v, w) \wedge \ell \in (g/B)'(v/B, w/B)$
4. $a' = a$

As operation symbol for the expansion we choose $\backslash$ since it is a kind of inverse operation of the quotient / (see also Theorem 6.3.2). An immediate conclusion is that under the above circumstances the expansion $(M/B)'\backslash B$ is a submodel of M. A more interesting and important fact is that the expansion is bisimilar to the submodel it is built from:

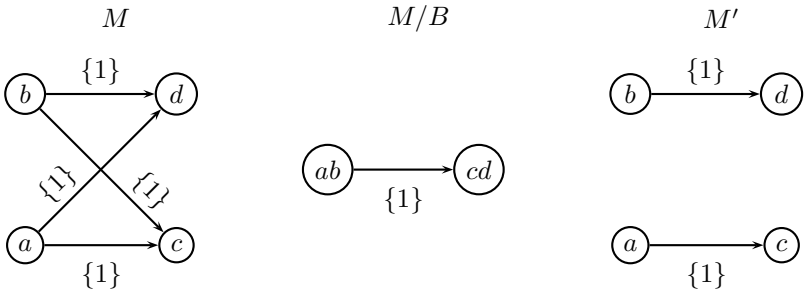


Figure 6.3: A model (left), a quotient (middle) and a possible expansion (right)

Theorem 6.3.1 *Let $M = (((V, E), g), a)$ be a model, B a bisimulation equivalence for M and $(M/B)' = ((((V/B)', (E/B)'), (g/B)'), (a/B)')$ a submodel of M/B. Then $(M/B)'$ and $(M/B)'\backslash B =_{df} (((V', E'), g'), a')$ are bisimilar.*

Proof: We claim that $\ni$ is a bisimulation between $(M/B)'$ and $(M/B)'\backslash B$.

So consider first arbitrary $v, w \in V$ and ℓ with $v \stackrel{\ell}{\rightarrow}_{g'} w$. This means in particular $(v, w) \in E'$, which implies $(v/B, w/B) \in E'$ (Point 2 of Definition 6.3.1). On the other hand,we have $\ell \in g'(v, w)$ and hence $\ell \in (g/B)'(v/B, w/B)$. Together this yields $v/B \stackrel{\ell}{\rightarrow}_{(g/B)'} w/B$, and because of $v \in v/B$ and $w \in w/B$ Item 1 of Definition 4.2.1 is fulfilled.

Conversely, let $v/B, w/B \in (V/B)'$ and ℓ be arbitrary with $v/B \stackrel{\ell}{\rightarrow}_{(g/B)'} w/B$. In particular, this means also $v/B \stackrel{\ell}{\rightarrow}_{(g/B)} w/B$. Now choose an arbitrary $v' \in v/B$. Analogously to the proof of Lemma 6.2.1 we can conclude the existence of $w' \in w/B$ with $v' \stackrel{\ell}{\rightarrow}_{g} w'$. Due to the definition of the expansion we have also $v' \stackrel{\ell}{\rightarrow}_{g'} w'$.

Since $a' = a$ holds Part 2 of Definition 6.1.1 is fulfilled for the same reasons as in Lemma 6.2.1. ■

Theorem 6.3.1 shows that the expansion of a submodel of a quotient and the submodel of the quotient itself are bisimilar. In general there may be more than one submodel of the original model which is bisimilar to a submodel of a quotient.

Figure 6.3 gives an example: in the middle we see the coarsest quotient M/B of the model M at the left. The model M' is a submodel of M which is bisimilar to M/B, as well as M itself (clearly, M/B is a submodel of itself). However, the expansion has a special property which singles it out between all other submodels bisimilar to the refined quotient: It is the greatest of all these submodels with respect to $\preceq$. This means that the expansion has an 'equivalent' dynamic behaviour as the submodel

it stems from and it has the maximal amount of 'desired' or 'useful' transitions. In [GMS11] the expansion operation was designed as a submodel with these properties. The next theorem states these considerations in an explicit manner:

Theorem 6.3.2 *Let $M = (((V, E), g), a)$ be a model, B a bisimulation equivalence for M and $(M/B)' = ((((V/B)', (E/B)'), (g/B)'), (a/B)')$ a submodel of the quotient M/B. Then $(M/B)'\backslash B =_{df} (((V', E'), g'), a')$ is the greatest submodel of M with respect to $\preceq$ that is bisimilar to $(M/B)'$ with $\ni$ as bisimulation witness. In particular, we have $(M/B)\backslash B = M$.*

Proof: By Theorem 6.3.1 $(M/B)'\backslash B \sim_{\ni} (M/B)'$ holds. So consider an arbitrary submodel $M'' = (((V'', E''), g''), a'')$ of $M = (((V, E), g), a)$ such that $M'' \sim_{\ni} (M/B)'$ holds. Let $(v'', w'') \in E''$ be an arbitrary edge of M'' and choose an arbitrary $\ell'' \in g''(v'', w'')$. First, because M'' is a submodel of M we also have $(v'', w'') \in E$. Second, due to $M'' \sim_{\ni} (M/B)'$ we have $(v''/B, w''/B) \in (E/B)'$ and $\ell'' \in (g/B)'(v''/B, w''/B)$. According to the definition of the expansion this leads to $(v'', w'') \in E'$ and $\ell'' \in g'(v'', w'')$ which finishes the proof. ■

We will use this characterisation in Chapter 12.

In some cases we will not only expand a submodel of the quotient but we will also do a similar operation on node labels. This is the content of the following definition:

Definition 6.3.2 *Let $M = (((V, E), g), a)$ be a model, B a bisimulation equivalence for M and $b : (V/B) \to L$ an arbitrary function. Then we define the* expansion *$(b\backslash B) : V \to L$ of b by $(b\backslash B)(v) = b(v/B)$.*

Note that $b\backslash B$ is well-defined because the equivalence classes of B are disjoint.

Chapter 7

Control of Plant Automata

The first class of problems we will apply our approach to are so-called plant automata, a concept closely related to Omega- and Büchi automata. We will see that they fit perfectly into our model-based framework. Our goal is to control a plant automaton, i.e., construct a refinement which guarantees a desired property. The properties we consider are closely related to ideas from the area of temporal logic. First we show how controllability can be decided using a quotient, and subsequently the construction of a control policy via a quotient construction.

7.1 Basic Definitions

The concept of a plant automaton was introduced in [MPS95] by the following definition:

Definition 7.1.1 (cited from [MPS95]) A *plant automaton* is a tuple $\mathcal{P} = (Q, \Sigma_c, \delta, q_0)$, where Q is a finite set of states, Σ_c is a set of controller commands, $\delta : Q \times \Sigma_c \to 2^Q$ is the transition function and $q_0 \in Q$ is an initial state.

In the further course of this paper there are some refinements and additional features of this concept. Here we will give a definition which captures and unifies all these

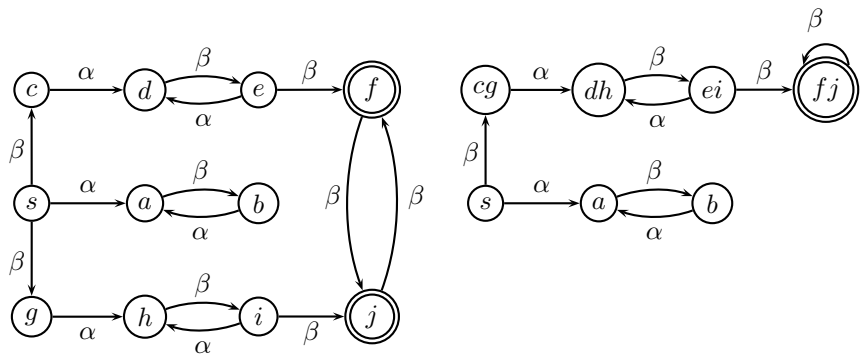


Figure 7.1: A plant automaton (left) and its coarsest quotient (right)

ideas in our model framework. From now on the term 'plant automaton' is understood in the sense of the following definition:

Definition 7.1.2 A *plant automaton* is a model $P = (((V, E), g), a)$ with finite label set L where a is a function $a : V \to 2^H$ with $H = \{\mathsf{init}, F\}$. Moreover, a has to satisfy $|\{v \,|\, \{\mathsf{init}\} \subseteq a(v)\}| = 1$.

Note that we now allow also an infinite number of states and nondeterminacy of transitions. However, it turns out that the problem investigated in [MPS95] can be handled also by our approach. Moreover, for an arbitrary plant automaton P and every bisimulation equivalence B for P the quotient P/B is again a plant automaton because the only real restriction $|\{v \,|\, a/B(v) = \mathsf{init}\}| = 1$ is clearly fulfilled. As a convention, we denote the unique node whose labelling contains init by s. In the sequel we will consider *traces* of a plant automaton. A *trace* of a plant automaton is an infinite walk in (V, E) with first node s.

As a convention for the graphic representation of a plant automaton we use double surroundings for all nodes in F. An example is given in the left part of Figure 7.1.

The goal is to *control* a plant automaton, i.e., construct a submodel with certain properties. We are not free to construct an arbitrary submodel but a submodel induced by a so-called *controller* (see Definition 7.1.4). In this submodel we have requirements to its traces. All its traces should fulfill properties with respect to F similar to concepts from temporal logic, as for example 'Every walk should infinitely

often visit a node with label F' or 'Every walk should eventually visit a node with label F'. If we associate the nodes in F with 'good' states of a system we can thereby describe that the refined automaton fulfills a desired safety ('The automaton is always in a good state') or liveness property ('Eventually something good will happen'). This concept is well known from the area of constructing and verifying temporal system properties as in [MP82, MP84, PV] and countless other publications.

We first define four predicates we will concentrate on:

Definition 7.1.3 Given a plant automaton $P = (((V, E), g), a)$ we define for all traces w of P the following family of predicates:

- $(F, \Box)_P(w) \overset{def}{\Leftrightarrow}$ all nodes of w bear label F
- $(F, \Diamond)_P(w) \overset{def}{\Leftrightarrow} w$ eventually visits a node with label F
- $(F, \Diamond\Box)_P(w) \overset{def}{\Leftrightarrow}$ eventually all nodes of w bear label F
- $(F, \Box\Diamond)_P(w) \overset{def}{\Leftrightarrow}$ infinitely many nodes of w bear label F

We will refer to these predicates also as *control objectives*. Clearly, the meanings and notions of these four predicates are inspired by modal and temporal logic (see e.g. [BA93, BAMP81, Pnu77]). In a slight notational abuse we say also that P has property ω if all its traces fulfill the predicate ω.

Next we want to specify how we can control a plant automaton. This will be done by choosing at every node $v \in V$ a value l from the label set L such that from v only edges with label l can be used. Intuitively, we have at every node a switch which enables different transitions, and we have to choose one switch position for every node. Formally, this is captured as follows:

Definition 7.1.4 Let $P = (((V, E), g), a)$ be a plant automaton. Then a *controller* for P is a function $C : V \to L$. Given a plant automaton P and a controller C for P, the plant automaton $P_C = (((V_C, E_C), g_C), a_C)$ is defined by

1. $V_C = V$ and $a_C = a$,
2. $(v_C, w_C) \in E_C \Leftrightarrow (v_C, w_C) \in E \wedge C(v) \in g(v_C, w_C)$
3. $g_C(v_C, w_C) = \{C(v_C)\}$

Clearly, P_C is a submodel of P, so this fits our framework without problems. Since in the sequel we want to reason about traces in a plant automaton we assume that for

every plant automaton under consideration and for every of its controllers every walk starting in s can be extended to an infinite one. The purpose of this stipulation is to avoid pathological cases where we have a universal quantifier over an empty set of infinite walks. Obviously, the plant automata from Figure 7.1 fulfil this requirement.

Given a plant automaton P our goal is to construct a controller C such that P_C fulfills some control objective property ω. If no such controller exists, this should be shown. This problem is referred to as the *controller synthesis problem*. We say that a plant automaton P is controllable *with respect to a property* ω if there is a controller C such that P_C has the property ω.

7.2 Deciding Controllability

7.2.1 Controllable Predecessors

We first sketch a family of algorithms for the plant automaton synthesis problem by means of fixpoint theory as for example in [TW91] or [MPS95]. These algorithms will only decide whether the controller synthesis problem is solvable, i.e., we will not yet show how to determine controllers. For this purpose we first introduce the concept of the set of *controllable predecessors* of a subset $V' \subseteq V$. Intuitively, this is the set of all nodes from which the plant automaton can be forced to enter some node in V' by a suitable controller (compare also e.g. [KPP09, HMCJF00, DRDW06]). Formally, this is defined as follows:

Definition 7.2.1 Let $P = (((V, E), g), a)$ be a plant automaton and consider a subset $V' \subseteq V$. The set $\pi(V')$ of *controllable predecessors* is defined as $\pi(V') = \{u \in V \mid \exists \ell \, \exists w : u \xrightarrow{\ell}_E w \wedge \forall v : (u \xrightarrow{\ell}_E v \Rightarrow v \in V')\}$.

Remark: The definition given in [MPS95] is slightly imprecise. There a state from which no transition with some label is possible belongs automatically to the set of controllable predecessors of every subset of V. This problem was fixed here by the existential quantifier for w. □

In the left plant automaton of Figure 7.1 we have e.g. $\pi(\{f\}) = \{e\}$ because we can force the automaton to choose the edge (e, f) if we use β as a control policy at node e. Similarly we have $\pi(\{a, j\}) = \{s, i\}$ (we choose α as control policy for s and β for i). However, note that $\pi(\{c\}) = \emptyset$ because neither by choosing α or β as control policy we can force a transition from any node to c.

The next lemma is about algebraic properties of the π-operator with respect to union and intersection. In particular, Part 2 shows the isotony of the π-operator with respect to set inclusion:

Lemma 7.2.1 *Let $P = (((V, E), g), a)$ be a plant automaton, and let $A, B \subseteq V$ be arbitrary subsets of V. Then the following inclusions hold:*

1. $\pi(A) \subseteq \pi(A \cup B)$.
2. $A \subseteq B \Rightarrow \pi(A) \subseteq \pi(B)$.
3. $\pi(A \cap B) \subseteq \pi(A) \cap \pi(B)$.
4. $\pi(A \cup B) \supseteq \pi(A) \cup \pi(B)$.

In general, the inclusions are strict.

Proof: First we show the inclusions; examples for strictness are given afterwards.
1: We fix an arbitrary $u \in \pi(A)$ and reason as follows:

$u \in \pi(A) \Leftrightarrow$
{ definition of π }
$\exists \ell\, \exists w\, \forall v : u \xrightarrow{\ell}_E w \wedge (u \xrightarrow{\ell}_E v \Rightarrow v \in A) \Rightarrow$
{ set theory, logic }
$\exists \ell\, \exists w\, \forall v : u \xrightarrow{\ell}_E w \wedge (u \xrightarrow{\ell}_E v \Rightarrow v \in (A \cup B)) \Leftrightarrow$
{ definition of π }
$u \in \pi(A \cup B)$

2: This follows directly from Part 1 due to $B = A \cup B$ for $A \subseteq B$.

3: Due to Part 2 we have $\pi(A \cap B) \subseteq \pi(A)$ and $\pi(A \cap B) \subseteq \pi(B)$. Then the claim follows by elementary set theory.

4: By Part 1 we have $\pi(A) \subseteq \pi(A \cup B)$ and $\pi(B) \subseteq \pi(A \cup B)$, so the claim follows by elementary set theory.

The strictness can be demonstrated on the left plant automaton of Figure 7.1. For Part 1 we note that $\pi(\emptyset) = \emptyset \subsetneq \{s\} = \pi(\emptyset \cup \{c, g\})$ holds. From this we obtain $\pi(\emptyset) \subsetneq \pi(\{c, g\})$ which provides an example for Part 2. The strict inclusion $\pi(\{d\} \cap \{f\}) = \pi(\emptyset) = \emptyset \subsetneq \{e\} = \{c, e\} \cap \{e, j\} = \pi(\{d\}) \cap \pi(\{f\})$ shows strictness for Part 3. Finally, we have $\pi(\{c\}) \cup \pi(\{g\}) = \emptyset \subsetneq \{s\} = \pi(\{c, g\}) = \pi(\{c\} \cup \{g\})$ for Part 4. ■

7.2.2 Algorithms for Deciding Controllability

All presented algorithms determine the set W of winning nodes, i.e., nodes from which a suitable refinement can enforce good behaviour in the sense of the predicate ω under consideration. Then the synthesis problem is solvable iff $q_0 \in W$ holds, and

control objective	fixpoint equation
$(F, \Box)$	$\nu W(F \cap \pi(W))$
$(F, \Diamond)$	$\mu W(F \cup \pi(W))$
$(F, \Diamond\Box)$	$\mu W \nu H(\pi(H) \cap (F \cup \pi(W)))$
$(F, \Box\Diamond)$	$\nu W \mu H(\pi(H) \cup (F \cap \pi(W)))$

Table 7.1: Control Objectives and Associated Fixpoint Equations

the synthesis can be done in a subsequent step. The set W can be characterised by fixpoint equations, depending on ω, as shown in Table 7.1. For the derivation of these equations consult again [TW91] or [MPS95].

For our purposes it will be useful to give explicit algorithms for the computation of these fixpoints. These algorithms can be obtained easily for the cases $(F, \Box)$ and $(F, \Diamond)$ by elementary fixpoint theory and lead to Algorithm 2 and 3 respectively.

Algorithm 2 Explicit Fixpoint Computation for the Case $(F, \Box)$

1: $W_0 \leftarrow V$
2: $i \leftarrow 0$
3: **repeat**
4: $\quad i \leftarrow i + 1$
5: $\quad W_i \leftarrow F \cap \pi(W_{i-1})$
6: **until** $W_i = W_{i-1}$
7: **return** W_i

Algorithm 3 Explicit Fixpoint Computation for the Case $(F, \Diamond)$

1: $W_0 \leftarrow \emptyset$
2: $i \leftarrow 0$
3: **repeat**
4: $\quad i \leftarrow i + 1$
5: $\quad W_i \leftarrow F \cup \pi(W_{i-1})$
6: **until** $W_i = W_{i-1}$
7: **return** W_i

Before introducing algorithms for the other properties we will take a closer look at Algorithms 2 and 3.

First we observe in Algorithm 3 that the equality $W_{i+1} = \bigcup_{j \leq i} \pi^i(F)$ holds (this can be

shown by simple induction). Symmetrically, in Algorithm 2 we have $W_{i+1} = \bigcap_{j \leq i} \pi^i(F)$. So the sequence W_0, W_1, W_2, ... is increasing in Algorithm 3 with respect to set inclusion, and decreasing in Algorithm 2. This ensures termination in the case of a finite node set V. Moreover, in Algorithm 3 the set W_{i+1} represents exactly the set of states from which the plant automaton can be forced by a suitable controller to enter a state with label F in at most i steps. Analogously, in Algorithm 2 the set W_{i+1} corresponds to the states from which the plant automaton can be forced to enter only states with label F for at least i steps.

The cases $(F, \Diamond\Box)$ and $(F, \Box\Diamond)$ are a little bit more complicated since we have to solve nested fixpoint iterations. This explains the nested loops in Algorithm 4 and its dual Algorithm 5.

Algorithm 4 Explicit Fixpoint Computation for the Case $(F, \Diamond\Box)$

1: $W_0 \leftarrow \emptyset$
2: **repeat**
3: $\quad H_0 \leftarrow V$
4: $\quad$ **repeat**
5: $\quad\quad H_{j+1} \leftarrow \pi(H_j) \cap (F \cup \pi(W_i))$
6: $\quad$ **until** $H_{j+1} = H_j$
7: $\quad W_{i+1} = H_j$
8: **until** $W_{i+1} = W_i$
9: **return** W_i

Algorithm 5 Explicit Fixpoint Computation for the Case $(F, \Box\Diamond)$

1: $W_0 \leftarrow V$
2: **repeat**
3: $\quad H_0 \leftarrow \emptyset$
4: $\quad$ **repeat**
5: $\quad\quad H_{j+1} \leftarrow \pi(H_j) \cup (F \cap \pi(W_i))$
6: $\quad$ **until** $H_{j+1} = H_j$
7: $\quad W_{i+1} = H_j$
8: **until** $W_{i+1} = W_i$
9: **return** W_i

We will give an exemplary execution of one of the algorithms immediately after the proof of Theorem 7.2.1.

7.2.3 Compatibility with Bisimulation Equivalences

We will now show that our generic approach works for the plant automaton synthesis problem. This is stated in the following theorem:

Theorem 7.2.1 *Let $P = (((V, E), g), a)$ be a plant automaton and B a bisimulation equivalence for P. If (P/B) is controllable with respect to a property ω in Ω then P is controllable with respect to ω.*

Proof: The idea of the proof is to execute the Algorithms 2 - 5 in parallel both on P and on P/B. For this purpose we denote the variables in the execution of the respective algorithm on P by W_i, Y_j and H_k, and in the execution on P/B by $(W/B)_i$, $(Y/B)_j$ and $(H/B)_k$. During this parallel execution we show inductively that in every step the equalities $W_i = \bigcup(W/B)_i$, $Y_j = \bigcup(Y/B)_j$ and $H_k = \bigcup(H/B)_k$ hold (remember again the writing conventions for the union of set systems from Chapter 2). Clearly, this holds for the initial assignment to variables indexed by 0 because of $\bigcup\emptyset = \emptyset$ and $\bigcup(V/B) = V$. Moreover, $\emptyset$ and V can be written as unions of suitable subsets of V/B. So we may assume as an induction hypothesis that $W_i = \bigcup(W/B)_i$, $Y_j = \bigcup(Y/B)_j$ and $H_k = \bigcup(H/B)_k$ hold before a new assignment, and that W_i, Y_j and H_k can be written as unions of suitable subsets of V/B.

If a simple assignment without new computation is done (Algorithm 4, Line 7 and Algorithm 5, Line 7) this preserves the claim in a trivial way. However, a closer look at the assignments with new computation (Algorithm 2, Line 5, Algorithm 3, Line 5, Algorithm 4, Line 5 and Algorithm 5, Line 5) shows that they are all of a form to which Theorem 6.2.1 is applicable. So after the execution of any new assignment the variables fulfill the equalities $W_{i+1} = \bigcup(W/B)_{i+1}$, $Y_{j+1} = \bigcup(Y/B)_{j+1}$ and $H_{k+1} = \bigcup(H/B)_{k+1}$. Hence at the end of the algorithm we have $\mathsf{init} \in W_{fin} \Leftrightarrow \mathsf{init} \in \cup(W/B)_{fin}$ (with the index fin we denote the sets returned by the respective computations). ■

We will illustrate both the function of Algorithm 4 and the previous proof on the plant automaton in Figure 7.1 and its coarsest quotient in the same Figure in Table 7.2.

The left column represents the values during the execution on the original automaton, the right one the execution on its coarsest quotient. The values of W_i and $(W/B)_i$, resp., denote the sets of nodes from which the automaton can be forced in $i-1$ steps to a node from which it can be forced to remain always in F. It is also instructive to see that in every line of this table the identities $W_i = \bigcup(W/B)_i$ and $H_j = \bigcup(H/B)_j$, resp., hold. This illustrates the idea behind the proof of Theorem 7.2.1.

$W_0 = \emptyset$	$(W/B)_0 = \emptyset$
$H_0 = V$	$(H/B)_0 = V/B$
$H_1 = \{f, j\}$	$(H/B)_1 = \{fj\}$
$H_2 = \{f, j\}$	$(H/B)_2 = \{fj\}$
$W_1 = \{f, j\}$	$(W/B)_1 = \{fj\}$
$H_0 = V$	$(H/B)_0 = V/B$
$H_1 = \{e, f, i, j\}$	$(H/B)_1 = \{ei, fj\}$
$H_2 = \{e, f, i, j\}$	$(H/B)_2 = \{ei, fj\}$
$W_2 = \{e, f, i, j\}$	$(W/B)_2 = \{ei, fj\}$
$H_0 = V$	$(H/B)_0 = V/B$
$H_1 = \{d, e, f, h, i, j\}$	$(H/B)_1 = \{dh, ei, fj\}$
$H_2 = \{d, e, f, h, i, j\}$	$(H/B)_2 = \{dh, ei, fj\}$
$W_3 = \{d, e, f, h, i, j\}$	$(W/B)_3 = \{dh, ei, fj\}$
$H_0 = V$	$(H/B)_0 = V/B$
$H_1 = \{c, d, e, f, g, h, i, j\}$	$(H/B)_1 = \{cg, dh, ei, fj\}$
$H_2 = \{c, d, e, f, g, h, i, j\}$	$(H/B)_2 = \{cg, dh, ei, fj\}$
$W_4 = \{c, d, e, f, g, h, i, j\}$	$(W/B)_4 = \{cg, dh, ei, fj\}$
$H_0 = V$	$(H/B)_0 = V/B$
$H_1 = \{s, c, d, e, f, g, h, i, j\}$	$(H/B)_1 = \{s, cg, dh, ei, fj\}$
$H_2 = \{s, c, d, e, f, g, h, i, j\}$	$(H/B)_2 = \{s, cg, dh, ei, fj\}$
$W_5 = \{s, c, d, e, f, g, h, i, j\}$	$(W/B)_5 = \{s, cg, dh, ei, fj\}$
$H_0 = V$	$(H/B)_0 = V/B$
$H_1 = \{s, c, d, e, f, g, h, i, j\}$	$(H/B)_1 = \{s, cg, dh, ei, fj\}$
$H_2 = \{s, c, d, e, f, g, h, i, j\}$	$(H/B)_2 = \{s, cg, dh, ei, fj\}$
$W_6 = \{s, c, d, e, f, g, h, i, j\}$	$(W/B)_6 = \{s, cg, dh, ei, fj\}$

Table 7.2: Example Execution of Algorithm 4

7.3 Implementation Details

In Section 7.4 we will show that the computation of a controller can be done via a bisimulation quotient (cf. also Algorithm 7 from Subsection 7.4.1). However, in order to see whether this approach can lead to a speed-up we first take a look at how to implement the actual computation of a controller.

7.3.1 Basic Computation of a Controller

In [MPS95] it is shown that Algorithms 2 - 5 are constructive in the sense that during their execution a controller can be determined.

The construction can be done as follows:

- If the sequence of W_i's is increasing with respect to set inclusion (as in Algorithm 2 and Algorithm 5) proceed as follows: Every time a state q is added to W_i for $i \geq 1$ (i.e., if $q \in W_{i+1} \backslash W_i$) there has to be an α such that $\delta(q, \alpha) \neq \emptyset$ and $\delta(q, \alpha) \subseteq W_i$ hold. Then we set $C(q) = \alpha$. For a node $v \in F$ we can choose an arbitrary value α with $|\delta(v, \alpha)| \neq 0$.

- In the case that the sequence of W_i's is decreasing with respect to set inclusion (this will happen at Algorithm 3 and Algorithm 4) the construction is symmetric to the above case: Every time if a state is kept in W_{i+1} (i.e., $q \in W_{i+1}$ holds) we choose an α with the properties $\delta(q, \alpha) \neq \emptyset$ and $\delta(q, \alpha) \subseteq W_i$, and set $C(q) = \alpha$ (clearly, it suffices to do this not until the last iteration because it makes no sense to control non-winning states).

Let us illustrate this method on the example execution of Algorithm 4 on the original plant automaton from Figure 7.1 (see the left column of Table 7.2): for all nodes in F (here f and i) we can choose an arbitrary controller which enables a transition. This forces us to choose $C(f) = C(i) = \beta$. The set difference $W_2 - W_1$ equals $\{e, i\}$, and we choose $C(e) = C(i) = \beta$ because of $\delta(e, \beta) \subseteq W_1$ and $\delta(i, \beta) \subseteq W_1$ (note that the choices $C(e) = \alpha$ and $C(i) = \alpha$ are not possible because of $\delta(e, \alpha) \subsetneq W_1$ and $\delta(i, \alpha) \subsetneq W_1$). For analogous reasons we obtain $C(d) = C(h) = \beta$, $C(c) = C(g) = \alpha$ and $C(s) = \beta$.

As we will see exemplary in Algorithm 6 these computations can be done during the execution of the respective algorithm without slowing down its asymptotical running time. So for the runtime discussions in the next subsection we note only that the running time of computing a controller for a given control objective is asymptotically the same as deciding controllability using one of Algorithms 2 - 5.

7.3.2 Running Time Considerations

There is some work about computing controllers as in the previous subsection (see e.g. [BL69, RW89, TW91, TW94]) but nowhere an analysis of the complexity of the presented algorithm is provided. As a first glance it looks like we could apply a stepping technique for computing the sequence of W_i's in Algorithm 2 and Algorithm 3 as used in a similar context for example in [Ehm03, EMS03a, EMS03b]. However, the predecessor operator lacks suitable algebraic properties for developing an analogous technique. In a relational setting, the properties used there can be written as $R;(S \cup T) = (R;S) \cup R;T$ and $(M \cup N)R = MR \cup NR$. For the π-operator we have only inclusions, which can also be strict as shown in Lemma 7.2.1.

First we propose in Algorithm 6 an implementation of Algorithm 3. This algorithm computes a controller simultaneously with the sets from Algorithm 3.

Let us take a closer look at this implementation:

- In Line 1 we start the iteration already with the set F instead of $\emptyset$. This is possible since in Algorithm 3 we have $W_1 = F$.
- For every node $v \in F$ we choose an arbitrary controller value (Lines 10 - 13), according to the discussion in Subsection 7.3.1.
- The controllable predecessors of W_i are determined pointwise, i.e., every node from a candidates set (see next item) is tested separately for membership in $\pi(W_i)$.
- We maintain a set V' in which we keep the candidates for all elements of potential predecessors (cf. Line 5 of Algorithm 3) to avoid unnecessary multiple computations. So if we look for possible predecessors of W_{i-1} (between Lines 17 and 27) it suffices to consider only nodes in V'.
- The nodes in $W_i - W_{i-1}$ are kept in a set ΔW and are added to W_{i-1} in Line 28. The reason for this is to obtain a exact implementation since we will represent all W_i's in one array (see the following discussion about the implementation of the set operations).
- If we find a node in $\pi(W_{i-1})$ (this happens if the condition in Line 19 evaluates to true) we add it to ΔW and remove it from V'. In this case we could immediately leave the loop beginning in Line 18 by a **break**-statement but we abstained from this possibility for the following reasons: first, a **break**-statement may lead to confusions while reading the code, and second it does not improve the running time in the worst case (an α with the property from Line 19 could be found in the very last pass through the loop starting in Line 18).

Algorithm 6 Implementation of Algorithm 3

1: $W_0 \leftarrow F$
2: $V' = V - F$
3: **for all** $v' \in V'$ **do**
4: determine $g_{out}(v')$
5: **end for**
6: $i \leftarrow 0$
7: **for all** $v \in V'$ **do**
8: $C(v) \leftarrow \mathsf{null}$
9: **end for**
10: **for all** $v \in F$ **do**
11: choose an arbitrary α such that $|\delta(v, \alpha)| \neq 0$
12: $C(v) \leftarrow \alpha$
13: **end for**
14: **repeat**
15: $i \leftarrow i + 1$
16: $\Delta W = \emptyset$
17: **for all** $v' \in V'$ **do**
18: **for all** $\alpha \in g_{out}(v')$ **do**
19: **if** $\delta(v', \alpha) \subseteq W_{i-1}$ **then**
20: $\Delta W = \Delta W \cup \{v'\}$
21: $V' = V' - \{v'\}$
22: **if** $C(v') \neq \mathsf{null}$ **then**
23: $C(v') \leftarrow \alpha$
24: **end if**
25: **end if**
26: **end for**
27: **end for**
28: $W_i = W_{i-1} \cup \Delta W$
29: **until** $W_i = W_{i-1}$

A tedious task is the implementation of the involved set operations. They can be done in the following manner:

- For the set V' we want a data structure which allows iteration through all elements of V' in $\mathcal{O}(V')$ time, and removal and insertion of an element in constant time. For this purpose we use the combination of a doubly linked list $\Lambda_{V'}$ of elements in V and an array $A_{V'}$ indexed by V. In $\Lambda_{V'}$ every element of V' occurs exactly one time. The array contains tuples consisting of a Boolean value and a pointer to an element of the doubly linked list. The Boolean part of $A_{V'}[v]$ equals true iff $v \in V'$ holds. In this case, the pointer of $A_{V'}[v]$ points to the corresponding element of $\Lambda_{V'}$; otherwise it is set to null. If we want to iterate over all elements of V' we use $\Lambda_{V'}$. Adding and removing an element from V' can be done in a similar manner as the analogous operations in the mixed representation of a set labelled graph (see Chapter 4).

- The set ΔW is represented by a linked list, consisting of elements from V, so iteration through all its elements can be done in linear time. The same holds for deletion of all its elements.

- The different sets W_i will be implemented by one Boolean array W with indices from V. To implement Line 1, we set $W[v] = \mathsf{false}$ for all $v \notin F$ and $W[v] = \mathsf{true}$ for all $v \in F$. In Line 28 we simply set $W[v]$ to true for all nodes in ΔW. This can not take place in Line 20 because this could change the value of W_{i-1}. In contrast, the described approach preserves the value of W_{i-1} till the end of the **until**-loop in Line 28.

To analyse the running time we assume that the plant automaton is given in mixed representation as described in Chapter 4. Together with the above considerations we obtain the following analysis:

- Lines 1 and 2 can be executed in $\mathcal{O}(|V|)$ time each. The same holds for the loop between Line 7 and 9.

- The determination of $g_{out}(v')$ for all $v' \in V'$ (Lines 3 - 5) can be done in $\mathcal{O}(|E| \cdot |L|)$ time. For this purpose we iterate for every $v' \in V'$ through the list $\Sigma[v']$ and add for every $w \in \Sigma[v']$ the values of $\Gamma[v'][w]$ to $g_{out}(v')$. The running time can be achieved if we maintain every $g_{out}(v')$ as a list and besides use during the iteration through $\Sigma[v']$ a Boolean array indexed by A where we record every value we added already to $g_{out}(v')$. We will see that this does not affect the asymptotic running time.

- The running time of the loop between Line 10 and 13 depends on Line 11. This line can be executed in constant time as follows: we first determine a node w with $(v, w) \in E$ (using a mixed representation we only have to deliver the first element of $\Sigma[v]$). Subsequently, we choose for α the first element of $\Gamma[v][w]$.
- The **until**-loop between Line 14 and Line 29 is executed at most $|V| + 1$ times because the sequence W_0, W_1, W_2, ... is strictly increasing with respect to $\subseteq$ (apart from the last iteration) and it is bounded from above by V.
- By construction, V' and W_i and hence ΔW and W_i are disjoint. So the loop condition in Line 29 reduces to the test $\Delta W = \emptyset$ which can be done in constant time.
- Line 16 can be done in constant time (we simply initialise a new doubly linked list) or in time $\mathcal{O}(|\Delta W|)$ (if we deallocate all list elements of ΔW). If we choose the second possibility we have an overall running time (summed up over all executions of the **until**-loop) of $\mathcal{O}(|V|)$ which will not affect the asymptotic running time.
- The **for**-loop between Line 17 and Line 27 is executed $|V'|$ times. In the worst case $|V'|$ equals 1 in Line 14 and is decremented by 1 in every run through the **until**-loop, so we can have a total of $\mathcal{O}(|V|^2)$ executions of this **for**-loop.
- The next **for**-loop from Line 18 till Line 26 is executed $|g_{out}(v')|$ times. Together with the previous result this leads to a total of $\mathcal{O}(|V|^2 \cdot |L|)$ executions.
- The comparison $\delta(v', \alpha) \subseteq W_{i-1}$ in Line 19 can be done in $\mathcal{O}(|\delta(v', \alpha)|)$ time (see Operation 1 of the mixed representation and the representation of W_{i-1} as a Boolean array).
- Every statement between Line 20 and Line 24 can be executed in constant time.
- The overall running time of Line 28 is in $\mathcal{O}(|V|)$; cf. the analysis of Line 16.

So the dominating parts are the computation of $g_{out}(v')$ for all $v' \in V - F$ (Line 3 - 5) with a running time in $\mathcal{O}(|E| \cdot |L|)$ and the **for**-loop from Line 18 till Line 26 with a running time in $\mathcal{O}(|V|^2 \cdot |L|)$. Because of $|E| \in \mathcal{O}(|V|^2)$ the running time of Algorithm 6 is in $\mathcal{O}(|V|^2 \cdot |L|)$. In symmetric manner we can implement Algorithm 2 which leads to the same running time.

The implementation of Algorithm 4 and 5 can be done by using implementations of Algorithm 2 and 3 as a subroutine for their loops from Line 5 till Line 7. The outer loop between Line 2 and Line 8 in both algorithms is executed in the worst case $\mathcal{O}(|V|)$ times. So the overall runtime is here in $\mathcal{O}(|V|^3 \cdot |L|)$.

7.4 Computing Controllers via Quotients

7.4.1 Compatibility of Controllers with Bisimulation Quotients

In Subsection 7.2.3 we showed that controllability can be decided with the use of bisimulation quotients. However, we did not investigate whether the actual computation of a controller is possible in an analogous manner. To our pleasure, this approach works with the help of the following theorem:

Theorem 7.4.1 *Let $P = (((V, E), g), a)$ be a plant automaton and B a bisimulation equivalence for P. Additionally, let C be a controller for P/B such that $(P/B)_C$ fulfills some property $\omega \in \Omega$. Then we have $(P/B)_C \backslash B = P_{C\backslash B}$. Moreover, $(P/B)_C \backslash B$ fulfills ω, too.*

Proof: In the sequel we use the following canonical notations, which we repeat explicitly to clarify matters:

- $P/B = (((V/B, E/B), g/B), a/B)$
- $(P/B)_C = ((((V/B)_C, (E/B)_C), (g/B)_C), (a/B)_C)$
- $(P/B)_C \backslash B = ((((V/B)_C \backslash B, (E/B)_C \backslash B), (g/B)_C \backslash B), (a/B)_C \backslash B)$
- $P_{C\backslash B} = (((V_{C\backslash B}, E_{C\backslash B}), g_{C\backslash B}), a_{C\backslash B})$

By definition we have $(V/B)_C \backslash B = V = V_{C\backslash B}$ and $(a/B)_C \backslash B = a = a_{C\backslash B}$, so it remains to show that $(E/B)_C \backslash B = E_{C\backslash B}$ and $(g/B)_C \backslash B = g_{C\backslash B}$ hold.

For the first equality we consider an arbitrary edge (v, w) in $(E/B)_C \backslash B$ and reason as follows:

$(v, w) \in (E/B)_C \backslash B \Leftrightarrow$
  { definition of expansion }
$(v, w) \in E \wedge (v/B, w/B) \in (E/B)_C \wedge$
$\exists \ell : \ell \in (g/B)_C (v/B, w/B) \wedge \ell \in g(v, w) \Leftrightarrow$
  { definition of $(E/B)_C$ }
$(v, w) \in E \wedge (v/B, w/B) \in (E/B) \wedge C(v/B) \in (g/B)(v/B, w/B) \wedge$
$\exists \ell : \ell \in (g/B)_C (v/B, w/B) \wedge \ell \in g(v, w) \Leftrightarrow$
  { definition of $(g/B)_C$ }
$(v, w) \in E \wedge (v/B, w/B) \in (E/B) \wedge C(v/B) \in (g/B)(v/B, w/B) \wedge$
$\exists \ell : \ell = C(v/B) \wedge \ell \in g(v, w) \Leftrightarrow$
  { C is a total function }
$(v, w) \in E \wedge (v/B, w/B) \in (E/B) \wedge C(v/B) \in (g/B)(v/B, w/B) \wedge$

$C(v/B) \in g(v,w) \Leftrightarrow$
{ Lemma 6.2.2, set theory }
$(v,w) \in E \wedge (v/B, w/B) \in (E/B) \wedge C(v/B) \in g(v,w) \Leftrightarrow$
{ Corollary 6.2.1, set theory }
$(v,w) \in E \wedge C(v/B) \in g(v,w) \Leftrightarrow$
{ definition of $C\backslash B$ }
$(v,w) \in E \wedge C\backslash B(v) \in g(v,w) \Leftrightarrow$
{ definition of $E_{(C\backslash B)}$ }
$(v,w) \in E_{(C\backslash B)}$

Now we know that $(E/B)_C\backslash B = E_{C\backslash B}$ holds, so that the equality $(g/B)_C\backslash B = g_{C\backslash B}$ is a simple implication of Part 3 of Definition 7.1.4 and the definition of $C\backslash B$ (cf. Definition 6.3.2).

As stated by Theorem 6.3.1, $(P/B)_C$ and $(P/B)_C\backslash B$ are bisimilar. So $(P/B)_C\backslash B$ fulfills ω, as an easy consequence of Corollary 6.1.1. ■

Theorem 7.4.1 leads immediately to Algorithm 7 which computes a controller for a given control property ω. In this algorithm, the implementations of Line 1 and Line 3 were already discussed. The actual problem, the computation of the controller in Line 2 was already investigated in Section 7.3.

Algorithm 7 Computing a Controller via the Coarsest Quotient

Require: a plant automaton $P = (((V,E),g),a)$ and a control property ω
1: compute the coarsest bisimulation B for P
2: compute a controller C for P/B such that $(P/B)_C$ fulfills ω
3: compute the expansion $(P/B)_C\backslash B$
Ensure: $(P/B)_C\backslash B$ fulfills ω

7.4.2 Running Time Considerations

The running times of $\mathcal{O}(|V|^2 \cdot |L|)$ and $\mathcal{O}(|V|^3 \cdot |L|)$ from Subsection 7.3.2 for computing controllers compared to the running time of $\mathcal{O}(|E| \cdot log|V| \cdot |L|)$ for computing the coarsest quotient gives hope to obtain a speed-up using Algorithm 7 instead of the immediate application of Algorithm 6 or related algorithms from Subsection 7.3.2. We will give an example of a family of plant automata with this property.

Consider the family $(P_i)_{i\in\mathbb{N}^+} = (((V_i,E_i),g_i),a_i)$ of plant automata, defined by

- $Q_i = \{s_i\} \dot\cup \{q_i^{jk} \mid 1 \leq j,k \leq i\}$

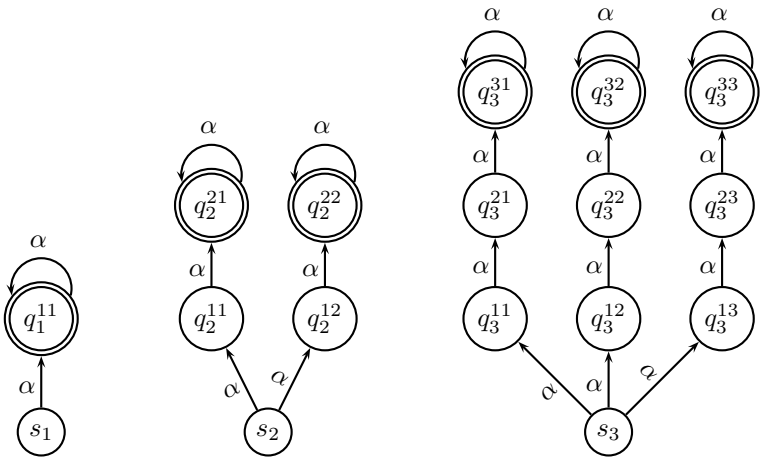


Figure 7.2: A Family of Plant Automata

- $E_i = \{(s_i, q_i^{1k}) \mid 1 \leq k \leq i\} \dot\cup \{(q_i^{j-1k}, q_i^{jk}) \mid 1 \leq k \leq i, 2 \leq j \leq i\} \dot\cup \{(q_i^{hi}, q_i^{hi}) \mid 1 \leq h \leq i\}$
- $g_i(e) = \alpha$ for all $e \in E_i$
- $a_i(v) = \begin{cases} \{\mathsf{init}\} & \text{if } v = s_i \\ \{F\} & \text{if } v \in \{q_i^{ij} \mid 1 \leq j \leq i\} \\ \emptyset & \text{otherwise} \end{cases}$

The first three members of this family P_1, P_2 and P_3 are depicted in Figure 7.2. Their coarsest quotients are sketched in Figure 7.3 (the nodes' captions are slightly simplified). Obviously, a controller C_i with respect to $(F, \Diamond)$ for every P_i is simply given by $C_i(v) = \alpha$ for every $v \in V_i$.

For the number of nodes and edges of P_i we have $|V_i| = i^2 + 1$ and $|E_i| = i^2 + i$, so $|E_i| \in \Theta(|V_i|) = \Theta(i^2)$. However, in the coarsest quotient P_i/B_i (where B_i is the coarsest bisimulation for P_i) we have for the number of nodes and edges the equalities $|V_i/B_i| = i + 1$ and $|E_i/B_i| = i + 1$ and hence $|E_i| \in \Theta(|V_i|) = \Theta(i)$.

Let us now consider the execution of Algorithm 6 on P_i. The Lines 1 - 13 can be executed in $\mathcal{O}(|E_i|) = \mathcal{O}(|V_i|)$ time. However, the bottleneck is the **until**-loop between Line 14 and Line 29. Before the first entry into this loop, W_0 is set to $\{q_i^{ij} \mid 1 \leq j \leq i\}$

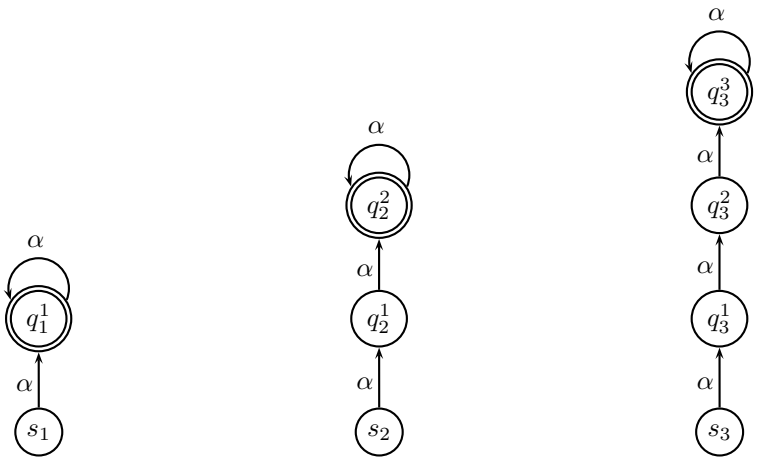


Figure 7.3: Coarsest Quotients of Plant Automata

in Line 2. After the first iteration we have $W_1 = \{q_i^{hj} \mid 1 \leq j \leq i, i-1 \leq h \leq i\}$, and inductively we get $W_h = \{q_i^{gj} \mid 1 \leq j \leq i, i-h \leq g \leq i\}$ for $0 \leq h \leq i-1$. Finally, this loop terminates after $i+1$ executions because of $W_{i+1} = W_i = V_i$. The set V' contains during the h-th execution $|V_i| - |W_h| = i^2 + 1 - hi$ elements for $1 \leq h \leq i$, so the loop between Line 17 and 17 has an overall running time of at least $\Theta(\sum_{h=1}^{i} i^2 + 1 - hi) = \Theta(\frac{1}{2}i(i^2 - i + 2)) = \Theta(i^3) = \Theta(|V_i|^{\frac{3}{2}})$.

Consider now the execution of Algorithm 7 on P_i where we implement Line 2 by Algorithm 6. The first line can be executed in $\mathcal{O}(|E_i| \cdot log(|V_i|)) = \mathcal{O}(|V_i| \cdot log(|V_i|))$ time. After this we obtain a quotient with $i+1$ nodes and edges as shown in Figure 7.3. Running Algorithm 3 on this quotient needs $\mathcal{O}((i+1)^2) = \mathcal{O}(|V_i|)$ time. Because Line 3 of Algorithm 7 can be executed in $\mathcal{O}(|V_i| + |E_i|) = \mathcal{O}(|V_i|)$ time, the overall running time is in $\mathcal{O}(|V_i| \cdot log(|V_i|))$ time. This is a speed-up compared to the immediate application of Algorithm 6 as analysed above. An even greater speed-up can be expected in an analogous situation for Algorithms 4 and 5.

Chapter 8

Target Models

This chapter is in a large part based on the work in [Glü11] and [Glü12]. We focus on optimality problems as shortest walks or maximum capacity walks and use an algebraic framework which captures a large variety of such problems. The goal is to refine a model in such a way that a distinguished set of nodes, the target set, is reached on an optimal walk. Before we investigate the use of bisimulation quotients for this kind of problem we discuss generally the possibility of refining such models and introduce some algorithms for obtaining optimality.

8.1 Dioids

To model the costs of a walk in a general manner we label the edges with elements drawn from a dioid. The use of dioids for this purpose is extensively described in [GM08b] and used for example in [BG09]. Similar approaches can be found in [BC75, Bou04, Min76]. Therefore, we will use the namings from [GM08b] although there are other namings for the same structures. In the further course we will deviate from [GM08b] since we concentrate on the refinement of models, which is not covered there.

Definition 8.1.1 A *complete dioid* is a structure $(D, \Sigma, 0, \cdot, 1)$ such that $(D, \sqsubseteq)$ is a complete lattice with supremum operator Σ and least element 0, where $\sqsubseteq$ is defined

by $x \sqsubseteq y \Leftrightarrow \Sigma\{x, y\} = y$, $(D, \cdot, 1)$ is a monoid and $\cdot$ distributes over Σ from both sides. $\sqsubseteq$ is called the *order* of the complete dioid.

The binary supremum operation in a complete dioid is denoted by $+$ and is referred to as *addition*, i.e. $x + y = \Sigma\{x, y\}$. Because 0 is the least element with respect to $\sqsubseteq$, we have $x + 0 = 0 + x = x$ for all $x \in D$. Moreover, $+$ is commutative, associative and idempotent (i.e., $x + x = x$ holds for all $x \in D$). The operation $\cdot$ is also called *multiplication*. Note that 0 is an annihilator of multiplication (i.e. $0 \cdot x = x \cdot 0 = 0$ for all $x \in D$) due to $\Sigma\,\emptyset = 0$. Often for readability the $\cdot$ is omitted, so ab stands for $a \cdot b$. As commonly known in this setting, both addition and multiplication are isotone with respect to the order, i.e. $a \sqsubseteq b$ implies $a + c \sqsubseteq b + c$ and $c + a \sqsubseteq c + b$ as well as $ac \sqsubseteq bc$ and $ca \sqsubseteq cb$ for all $a, b, c \in D$. We use $a \sqsubset b$ as an abbreviation for $a \sqsubseteq b \wedge a \neq b$, and the signs $\sqsupseteq$ and $\sqsupset$ instead of $\sqsubseteq^\circ$ and $\sqsubset^\circ$ respectively. In order to avoid notational overflow will use D to denote both the carrier set of a complete dioid and the complete dioid itself when the meaning is clear from the context.

In our setting the supremum operation models choice, and the multiplication models composition off walks. So if we consider to walks we can decide which of them is the better one (or they may both be even attractive to us) with respect to our optimality objective. This motivates the use of selective dioids. A complete dioid is called *selective* if $a + b \in \{a, b\}$ holds. Obviously this can be extended to the suprema of arbitrary nonempty finite sets. In this case, $\sqsubseteq$ is a linear relation. In the sequel we will consider mainly selective complete dioids; we call them *s-dioids* for short.

There are many examples for complete dioids, such as $(\mathbb{R} \cup \{-\infty, \infty\}, \sup, -\infty, \inf, \infty)$ or $(2^{\mathbb{N}}, \cup, \emptyset, \cap, \mathbb{N})$. The lattice order is $\leq$ in the first example and $\subseteq$ in the second one. The first one is also an s-dioid, whereas the second one is not.

As a special class of complete dioids we consider *cumulative* dioids, which are characterised by $a \sqsubseteq 1$ for all a, i.e., 1 is the greatest element with respect to the dioid's order. This includes the most used and common dioids, for example the well-known sup-inf dioid $(\mathbb{R} \cup \{-\infty, \infty\}, \sup, -\infty, \inf, \infty)$ is cumulative. In the context of language analysis they are also used under the name *1-bounded* in [EKL08]. We can characterise cumulative dioids in different equivalent ways, as stated in the following lemma:

Lemma 8.1.1 *The following statements are equivalent:*

1. $(D, \sum, 0, \cdot, 1)$ *is a cumulative dioid.*
2. *For all* $a, b, c \in D$ *the implications* $a \sqsubseteq b \Rightarrow ac \sqsubseteq b$ *and* $a \sqsubseteq b \Rightarrow ca \sqsubseteq b$ *hold.*
3. *For all* $a, b \in D$ *the inequalities* $ab \sqsubseteq a$ *and* $ba \sqsubseteq a$ *hold.*

Proof: $1 \Rightarrow 2$: Let $a, b, c \in D$ be arbitrary with $a \sqsubseteq b$. Because of isotony of multiplication with respect to. $\sqsubseteq$ and the assumption $c \sqsubseteq 1$ we have $ac \sqsubseteq a \cdot 1 = a$ and hence $ac \sqsubseteq b$. The other implication is shown analogously.
$2 \Rightarrow 3$: For arbitrary $a, b \in D$ we have $a \sqsubseteq a$ and due to b) we have $ab \sqsubseteq a$ (choose $a := a$, $b := a$ and $c := a$). The other inequality follows analogously.
$3 \Rightarrow 1$: In 3 we chose an arbitrary b and set $a = 1$. ■

The meanings and interpretations of these equivalent characterisations will become clear in the next section.

8.2 Models and Costs

8.2.1 Costs, Distances and Optimal Walks

As mentioned above, we will investigate transition systems with costs for each transition and target sets which are to be be reached. This is subsumed in the next definition in the context of models:

Definition 8.2.1 A *target model* is a model $M = (((V, E), g), a)$ with the following properties:

1. g is a mapping from E into the power set of the carrier set of an s-dioid $(D, \Sigma, 0, \cdot, 1)$ with $g(v, w) \neq \emptyset$ for all $(v, w) \in E$.

2. a is a mapping from V into the set $\{\mathsf{trans}, \mathsf{fin}\}$.

3. From every node $v \in V$ with $a(v) = \mathsf{trans}$ some node $w \in V$ with $a(w) = \mathsf{fin}$ is reachable.

4. Every node $v \in V$ with $a(v) = \mathsf{fin}$ has outdegree zero.

With the above notations, the dioid $(D, \Sigma, 0, \cdot, 1)$ is called the *associated* dioid of M. Note that the dioid itself is not determined by the model (only its carrier set), but we will tacitly assume that the dioid is clear from the context.

The function g models the cost of a transition and is hence called the *cost function* of M. The nodes in the set $\{v \in V \mid a(v) = \mathsf{fin}\}$ model the states we want to each from any node outside on an optimal walk. So the set $\{v \in V \mid a(v) = \mathsf{fin}\}$ is called the *target set* of M. As abbreviation for the target set we use the notation V_T. With this interpretation Part 4 of the definition makes sense since we do not want to exit from a node in the target set.

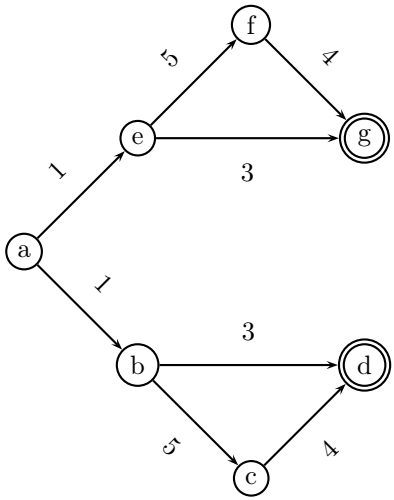


Figure 8.1: A Target Model

In practice, target models are in the most cases labelled. We dropped this requirement here, because the coarsest quotient of a uniquely labelled model need not to be uniquely labelled itself, as pointed out in section 6. So Definition 8.2.1 can serve as a consistent framework for both target models derived from practical applications and their quotients.

In the latter course we will deal with systems which are no 'perfect' target models in intermediate steps of an algorithm which constructs target models. So a *defect target model* fulfills the Parts 1, 2 and 4 of Definition 8.2.1, but not necessarily Part 3. For a (possibly defect) target model $M = (((V, E), g), a)$ and a subgraph $G' = (V', E') \preccurlyeq (V, E)$ the *restriction* of M by G', denoted by $M|_{G'}$, is defined by $M|_{G'} = (((V', E'), g|_{E'}), a|_{V'})$.

As a drawing convention for a graphical representation we declare that nodes in the target set are doubly surrounded and all other nodes are simply surrounded. If the target model is uniquely labelled we omit the set braces at the edge captions. The same holds for the node captions in quotients. So Figure 8.1 shows a uniquely labelled target model with target set $\{d, g\}$, but it is not yet clear what the associated s-dioid is because the operations on the dioid elements are not defined. This will have to be done in the context. We will refer to this later.

As already mentioned, we will use the values from the associated s-dioid to generalise costs of walks. To this purpose we introduce the following definition:

Definition 8.2.2 Let $M = (((V, E), g), a)$ be a target model and $(D, \Sigma, 0, \cdot, 1)$ the associated s-dioid. Then for a walk $w = x^1 x^2 \dots x^n$ in G the *cost* $c(w)$ of w is defined by $c(w) = \sum_{lab \in L(w)} \prod_{i=1}^{n-1} lab^i$. For two nodes x and y the *distance* $d(x, y)$ between x and y is defined by $d(x, y) = \sum_{w \in W(x,y)} c(w)$. In a target model the *target distance* $d(x)$ of a node x is defined as $d(x) = \sum_{t \in V_T} d(x, t)$. A walk $w = x^1 x^2 \dots x^n$ is called *optimal* if the equality $c(w) = d(x^1, x^n)$ holds.

This is a generalisation of the definitions of shortest walks or maximum capacity walks. The cost of a walk in these classic examples is defined as the sum and the maximum, resp, of the edge labels along it. Because we allow set-valued edge labels we take the optimum (here denoted by the sum) of all its labellings (note that sum and maximum correspond to multiplication in the s-dioid corresponding to shortest walks and maximum capacity walks). The distance of to nodes is defined as the optimum of the costs of all walks (this optimum corresponds to the infimum and the supremum in the case of shortest and maximum capacity walks, resp.).

Note that the cost of a walk and the distance between two nodes as well as the target distance of a node itself are always well-defined, because in an s-dioid suprema of arbitrary sets always exist. The distance $d(v, w)$ of two nodes v and w such that w is not reachable from v is defined as the supremum of the empty set and hence equals 0. Also in general $d(v, w) = d(w, v)$ does not hold. In a uniquely labelled target model the obvious equality $c(w) = \prod_{i=1}^{n-1} l(w)$ holds. Optionally, if we deal with various target models at the same time, we equip the symbols for cost and distance with an index indicating the target model under consideration.

Because the empty word is contained in the set of labellings of the walks from a node to itself the distance $d(v, v)$ between a node and itself equals the greatest element of the associated dioid. In the case of a cumulative s-dioid we have $d(v, v) = 1$. Due to Definition 8.2.1 nodes in the target set do not have outgoing edges, so the only labelling from such a node to itself is the empty word. Hence $d(v)$ equals the greatest element of the associated dioid for all nodes v in the target set.

If we choose the s-dioid $(\mathbb{R}_0^+ \cup \{\infty\}, \inf, \infty, +, 0)$ for the edge labels in a uniquely labelled target model, the cost of a walk corresponds to its length in its classic sense as the sum of the weights of its edges. The distance of two nodes corresponds to the length of a shortest walk connecting these two nodes, and the target distance $d(x)$ in

this setting to the minimal length of a shortest walk from x leading into the target set. Note that the order $\sqsubseteq$ in this s-dioid corresponds to $\geq$: if we have the choice between two walks we will choose the one with the lower cost, so lower numbers are more desirable than greater ones. Similarly, if the s-dioid $(\mathbb{R} \cup \{-\infty, \infty\}, \sup, -\infty, \inf, \infty)$ is chosen, the cost corresponds to the capacity (i.e., the minimum of all labels along a walk) of a walk, and the distance $d(x, y)$ to the maximum capacity of all walks in $W(x, y)$ (note that here the order $\sqsubseteq$ corresponds to $\leq$).

8.2.2 Properties and Existence of Optimal Walks

In the following we will investigate some properties of optimal walks. Some of them are important in the sequel, others are without further consequences but nevertheless of interest.

A simple, but for further considerations important, observation is stated in the next lemma:

Lemma 8.2.1 *Let w be an optimal walk from x to y, and let z be an inner node of w. Then $c(w) = d(x, z) \cdot d(z, y)$ holds.*

Proof: Denote by w_x a subwalk of w leading from x to z, and chose w_y so that $w = w_x \bowtie w_y$ holds. Moreover, let w_{xz} be an optimal walk from x to y and w_{zy} be an optimal walk from z to y. The inequality $c(w) \sqsubseteq d(x, z) \cdot d(z, y)$ can be shown as follows:

$c(w) =$
  { splitting of w }
$c(w_x) \cdot c(w_y) \sqsubseteq$
  { w_{xz} and w_{yz} are optimal }
$c(w_{xy}) \cdot c(w_{yz}) =$
  { Definition 8.2.2 }
$d(x, z) \cdot d(z, y)$

The other inequality $d(x, z) \cdot d(z, y) \sqsubseteq c(w)$ is also not hard to show:

$d(x, z) \cdot d(z, y$
  { Definition 8.2.2 }
$c(w_{xy}) \cdot c(w_{yz}) =$
  { gluing of walks }
$c(w_{xy} \bowtie w_{yz}) \sqsubseteq$
  { $w_{xy} \bowtie w_{yz} \in W(x, y)$, w optimal }
$c(w)$

These two inequalities show the desired equality. ■

Note that contrary to the intuition based on the shortest path problem an optimal walk need not be the gluing of two optimal walks. To see this take a look at the target model in Figure 8.1, and interpret the edge labels in the s-dioid $(\mathbb{R} \cup \{-\infty, \infty\}, \sup, -\infty, \inf, \infty)$: Then aeg is an optimal walk whereas its subwalk eg is not.

Even a weaker version of the above observation is not correct. There is a target model $M = (((V, E), g), a)$ with associated cumulative s-dioid and two connected nodes v^1 and v^2 with the the following property: for every optimal walk w in $W_{(V,E)}(v, w)$ there is a non-optimal subwalk w' of w. An example is given by the following uniquely labelled target model, sketched in Figure 8.2:

- $V = \{s, v, t\} \cup \bigcup_{n \in \mathbb{N}} \{v^n\}$
- $E = \{(s, v)\} \cup \bigcup_{n \in \mathbb{N}} \{(v, v^n), (v^n, t)\}$
- $g(s, v) = \{1\}$
- $g(v, v^n) = g(v^n, t) = \{2 - 2^{-n}\}$ for all $n \in \mathbb{N}$
- $a(t) = \mathsf{fin}$
- $a(v) = \mathsf{trans}$ for all $v \in V - \{t\}$

If we interpret the edge labels in the cumulative s-dioid $(\mathbb{R} \cup \{-\infty, \infty\}, \sup, -\infty, \inf, \infty)$ every walk from s to t is an optimal one, containing a subwalk from v to t. But there is no optimal walk at all from v to t because we have $d(v, t) = 2$ and $c(w) \neq 2$ for all $w \in W_G(v, t)$.

This shows also that in general an optimal walk need not exist. However, if the set of edge labels is finite we are in a much better situation. The key insight is the following theorem which was proven already back in 1952 (Theorem 4.4 in [Hig52]; in language theory the subject of this theorem is known as *scattered subword*, cf. [FN08, Sal03]):

Theorem 8.2.1 *If X is any set of words formed from a finite alphabet, it is possible to find a finite subset X_0 of X such that, given a word w in X, it is possible to find w_0 in X_0 such that the letters of w_0 occur in w in their right order, though not necessarily consecutively.*

This theorem is a valuable ingredient for the proof of the following theorem:

Theorem 8.2.2 *Let $M = (((V, E), g), a)$ be a target model with finite label set L and associated cumulative s-dioid $(D, \sum, 0, \cdot, 1)$. Then for every $v \in V$ there is a walk w from v into the target set with $c(w) = d(v)$.*

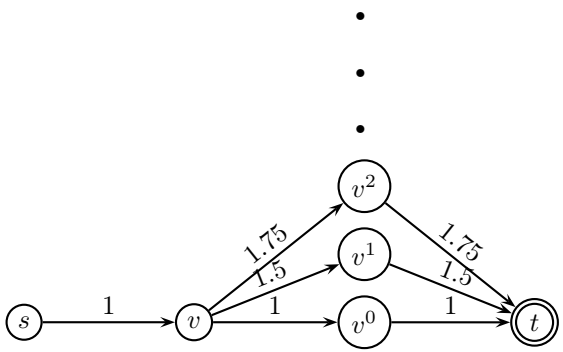


Figure 8.2: Walks without Optimal Subwalks

Proof: Let $M = (((V, E), g), a)$, L and $(D, \sum, 0, \cdot, 1)$ be as above, and fix an arbitrary $v \in V$. Then the set of all labellings L_v of walks from v into the target set is a language over L, and due to Theorem 8.2.1 there is a finite subset $L'_v \subseteq L_v$ such that for every $l_v \in L_v$ there is an $l'_v \in L'_v$ such that the symbols of l'_v occur in the same order also in l_v. For an arbitrary $l = \ell^1\ell^2 \dots \ell^n \in L^*$ we define the cost of l by $c(l) =_{df} \prod_{i=1}^{n} \ell^i$. By definition, we have $d(v) = \sum\{c(l_v) \mid l_v \in L_v\}$. However, due to the above observation, isotony of multiplication and cumulativity, for every $l_v \in L_v$ there is an $l'_v \in L'_v$ with the property $c(l_v) \sqsubseteq c(l'_v)$. Hence the target distance of v can be written as $d(v) = \sum\{c(l'_v) \mid l'_v \in L'_v\}$. Because L'_v is finite and $\sqsubseteq$ is a linear order there is a $l'_v \in L'_v$ with $d(v) = c(l'_v)$. But by construction, there is also a walk w from v into the target set of M with $l'_v \in l(w)$ and hence $d(v) = c(w)$. ■

8.2.3 Label-optimised Models

Before applying some algorithm to refine target models we have to deal with the fact that the associated graph need not to be uniquely labelled. We will here consider only the case of finitely labelled target models. As it will turn out, one of the labels suffices to generate a target model with a, for our purposes, sufficiently similar behaviour. This is done in the following definition:

Definition 8.2.3 Let $M = (((V, E), g), a)$ be a finitely labelled target model and

$(D, \Sigma, 0, \cdot, 1)$ the associated s-dioid. Then the *label-optimised* target model $M_o = (((V_o, E_o), g_o), a_o)$ is defined as follows:

1. $g_o(v, w) = \{ \sum g(v, w) \}$ for all $(v, w) \in E$
2. $V_o = V$, $E_o = E$ and $a_o = a$

Because M is finitely labelled we have $g_o(v, w) \subseteq g(v, w)$ for all $(v, w) \in E$. Together with part b) from the definition this means that M_o is a submodel of M.

The construction of the label-optimised model is justified by the following lemma which states the above mentioned similarity explicitly in a formal way:

Lemma 8.2.2 *Let $M = (((V, E), g), a)$ be a target model and $M_o = (((V_o, E_o), g_o), a_o)$ its label-optimised target model. Then for all $x, y \in V$ the distance between x and y in M equals the distance between x and y in M_o.*

Proof: From the definition of the label-optimised model it is obvious that every walk in M also exists in M_o and vice versa. So we assume w.l.o.g. that y is reachable from x both in M and M_o (otherwise the distance between x and y equals 0 both in M and M_o, and we are done). Now it suffices to show that for every walk from x to y in M the costs in M and M_o are equal. So let $w = w_1 w_2 \dots w_n$ be a walk in M from x to y, denote its cost in M by $c_M(w)$ and in M_o by $c_{M_o}(w)$ and its set of labellings in M by $L_M(w)$ and its labelling in M_o by $l_{M_o}(w)$. Due to construction of M_o we have $L_M(w) \ni l_{M_o}(w)$ and hence $c_M(w) \sqsupseteq c_{M_o}(w)$. On the other hand, we have $l^i_M \sqsubseteq l^i_{M_o}$ for all $l_M \in L_M(w)$ and $i \in \{1..n-1\}$. Isotony of multiplication yields $\prod_{i=1}^{n-1} l^i_M \sqsubseteq \prod_{i=1}^{n-1} l^i_{M_o}$ for all $l_M \in L_M(w)$, so we also have the converse inequality $c_M(w) \sqsubseteq c_{M_o}(w)$. ■

So if we want to compute the distances in a target model it suffices to compute the distances its label-optimised model.

8.3 Optimality and Refineability

To make things easier it turns out to be useful to consider first uniquely labelled target models. The obtained results will hold in a similar formulation for general target models due to Lemma 8.2.2.

Until now we did not make statements about the existence of optimal walks. E.g., in a uniquely labelled target model with label set $\mathbb{R}$ there is no shortest walk between two nodes in the presence of a cycle with negative length (by definition, in this context

the distance between two nodes can be $-\infty$, but there is no walk with this cost). If the unique labels are drawn from a cumulative s-dioid and the target model is finite, we are in a much better position: We can show that between two reachable nodes an optimal walk always exists, and moreover that there is even a path with optimal costs between two reachable nodes. This is the content of the following lemma.

Lemma 8.3.1 *Let $M = (((V, E), g), a)$ be a finite uniquely labelled target model with a cumulative associated s-dioid $(D, \sum, 0, \cdot, 1)$, and let x and y be two reachable nodes. Then there is a path $p \in P(x, y)$ in (V, E) with $c(p) = d(x, y)$.*

Proof: Let $x, y \in V$ be arbitrary reachable nodes and let $w \in W(x, y)$ be an arbitrary walk in (V, E). Now we assume that w contains a repeated node, i.e. $w = x^1 x^2 \dots x^i \dots x^j \dots x^n$ with $x^i = x^j$ and $i \neq j$. Consider now the walk $w' = x^1 x^2 \dots x^{i-1} x^j \dots x^n$ from x^1 to x^n. Due to cumulativity we have $c(x^1 x^2 \dots x^i \dots x^j) \sqsubseteq c(x^1 x^2 \dots x^{i-1})$, and together with the isotony of multiplication we obtain $c(w) \sqsubseteq c(w')$. If we repeat this construction sufficiently (but finitely!) often we obtain from w a path $p \in P(x, y)$ in (V, E) with $c(w) \sqsubseteq c(p)$. Therefore in this case $d(x, y) = \sum_{w \in W(x,y)} c(w) = \sum_{p \in P(x,y)} c(p)$ holds. Since M is supposed to be finite there are only finitely many paths from x to y and hence there is a $p \in P(x, y)$ with $c(p) = d(x, y)$ (note that the order in an s-dioid is linear, so every finite set contains a least element). ■

We want to ensure that every walk from a node outside the target set leading into the target set is an optimal one. To this purpose we will not only remove 'bad' edges bad also 'bad' edge labels. This goal will be achieved by constructing a suitable target submodel, which motivates the following definition (which is valid for the case of general target models):

Definition 8.3.1 For a target model $M = (((V, E), g), a)$ a target submodel $M' \preceq M$ is called an *optimal target submodel*, if for all walks w in M' from x to any node t with $a(t) = \mathsf{fin}$ the cost $c_{M'}(w)$ equals the target distance $d_M(x)$ in M (note that a target submodel is also a target model and therefore in an optimal target submodel the target set has to be reachable from every node outside of it).

In our example from Figure 8.1 the labels can be interpreted in the s-dioid $(\mathbb{R} \cup \{-\infty, \infty\}, \inf, \infty, +, 0)$. An optimal target submodel in this case is given in Figure 8.3. Assuming the labels to represent elements of the s-dioid $(\mathbb{R} \cup \{-\infty, \infty\}, \sup, -\infty, \inf, \infty)$ we see an optimal target submodel in Figure 8.4.

An immediate consequence of the Lemmata 8.2.2 and 8.3.1 and Definition 8.3.1 is the following corollary:

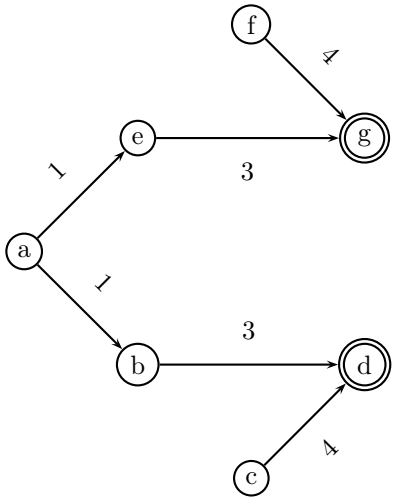


Figure 8.3: An Optimal Target Submodel

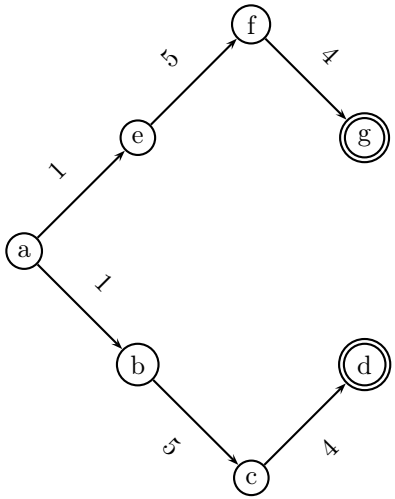


Figure 8.4: Another Optimal Target Submodel

Corollary 8.3.1 *Let M be a finitely labelled target model, M_o its label-optimised target model and M'_o an optimal target submodel of M_o. Then M'_o is an optimal target submodel of M, too.*

8.4 Target Models with Cumulative S-Dioids

The shortest walk problem in presence of negative cycles shows that not every target model (not even every uniquely labelled target model) has an optimal submodel. Target models, which have an optimal submodel, are called *refineable*. We will show that cumulativity is a necessary and sufficient condition for an s-dioid to guarantee the refineability of target models with it as associated s-dioid:

Theorem 8.4.1 *Let $(D, \sum, 0, \cdot, 1)$ be an s-dioid. Then every finite target model with D as associated s-dioid is refinable iff D is cumulative.*

Proof: In order to show the direction $\Rightarrow$ we consider Algorithm 8, which is basically a variant of Dijkstras algorithm for shortest paths (note that this algorithm works only on uniquely labelled target models). The correctness of this algorithm is shown in a way very similar to the known arguments for the common Dijkstra algorithm (see e.g. [Dij] and [Jun05]). In contrast to there, the edges are labelled with elements of a general cumulative s-dioid instead of $(\mathbb{R}_0^+ \cup \{\infty\}, \inf, \infty, +, 0)$, and we are not interested in computing shortest paths from a single node to all other nodes but we want to determine optimal walks leading into a given target set from every node outside of this target set. A feature shared with the classic case is the fact that in both cases prolonging a walk can not improve its cost. So for every walk $v^1 v^2 \dots v^n v^{n+1}$ we have the inequality $c(v^1 v^1 \dots v^n) \sqsupseteq c(v^1 v^2 \dots v^{n+1})$ due to the definition of the cost and part c) of Lemma 3.

First, we show that the algorithm terminates. The while-loop terminates because we can use $|U|$ as a termination function: $|U|$ is increased by one in every run through the loop and it is bounded from above by $|V|$. $|V|$ itself is finite because M is assumed to be finite. The for-loops terminate since they run over finite sets only.

To prove the correctness we will reason about intermediate results of the algorithm. Therefore we introduce the concept of a *maximal path* starting from a node with respect to. a given successor array. An array succ as in the above algorithm can be interpreted as a the encoding of a function $\mathsf{succ} : V \to V \dot\cup \{\mathsf{null}\}$ which assigns every node to at most one successor. In an obvious manner this defines a graph $G_{\mathsf{succ}} = (V_{\mathsf{succ}}, E_{\mathsf{succ}})$ with $V_{\mathsf{succ}} = V$ and $(v^1, v^2) \in E_{\mathsf{succ}}$ iff $\mathsf{succ}(v^1) = v^2$. Then the maximal path $mp(v, \mathsf{succ})$ is the maximal (i.e. not prolongable) walk (sic!) starting in v in G_{succ}. Intuitively, $mp(v, \mathsf{succ})$ starts in v and follows the instructions encoded

Algorithm 8 Dijkstra-like Algorithm in Case of Edge Labels from a Cumulative S-Dioid

Require: $M = (((V, E), g), a)$ is a finite uniquely labelled target model with cumulative associated s-dioid $(D, \sum, 0, \cdot, 1)$

initialise dist as an array with indices from V and values from D
initialise succ as an array with indices from V and values from V
initialise U as set with elements from V
for all $t \in V$ with $a(t) = \mathsf{fin}$ **do**
    $\mathsf{dist}[t] \leftarrow 1$
    $\mathsf{succ}[t] \leftarrow \mathsf{null}$
end for
for all $v \in V$ with $a(v) = \mathsf{trans}$ **do**
    choose a $t \in V_T$ with $\ell(v, t) = \sum_{t' \in V_T} \ell(v, t')$
    $\mathsf{dist}(v) \leftarrow \ell(v, t)$
    $\mathsf{succ}(v) \leftarrow t$
end for
$U \leftarrow V_T$
while $U \neq V$ **do**
    choose $v \notin U$ arbitrarily with $\mathsf{dist}(v) = \sum_{v' \notin U} \mathsf{dist}(v')$
    $U \leftarrow U \cup \{v\}$
    for all $(u, v) \in E$ with $u \notin U$ **do**
        if $g(u, v) \cdot \mathsf{dist}(v) \sqsupset \mathsf{dist}(u)$ **then**
            $\mathsf{dist}(u) \leftarrow g(u, v) \cdot \mathsf{dist}(v)$
            $\mathsf{succ}(u) \leftarrow v$
        end if
    end for
end while

Ensure: $\forall v \in V : \mathsf{dist}(v) = d(v)$, succ encodes an optimal submodel of M

in succ till it reaches a node without any further successor. Note that this definition makes sense only if succ encodes an acyclic graph.

The proof is done via the combined invariant $I \equiv I_1 \wedge I_2 \wedge I_3$ for the while-loop, defined by:

1. $I_1 \equiv \forall u \in U : \mathsf{dist}(u) = d(u)$,
2. $I_2 \equiv \forall u \in U : c(mp(u, \mathsf{succ})) = \mathsf{dist}(u)$, and
3. $I_3 \equiv \forall u \notin U : \mathsf{dist}(u) = d(u)$ in $M|_{U \cup \{u\}}$.

I_1 states that all nodes in the set U are already correctly labelled, i.e., the value of d coincides with the one of dist on all nodes of U. According to I_2, succ encodes optimal walks starting in any node in U. For nodes outside of U, by I_3 we know at least that dist gives the target distance in $M|_{U \cup \{u\}}$, i.e., $\mathsf{dist}(u)$ equals the optimal cost of a walk from u into V_T whose inner nodes are all in U. Clearly, after the **while**-loop we have $U = V$ which implies the ensured properties due to I_1 and I_2.

Now we have to show that I is indeed an invariant of the **while**-loop. It clearly holds after the assignment in Line 13 due to the assignments made during the preceding **for**-loops. Let now v be the node chosen by the algorithm in Line 15. Because U is modified in Line 16 we write $U \cup \{v\}$ and $U - \{v\}$ to avoid misunderstandings, even if this notation may be redundant.

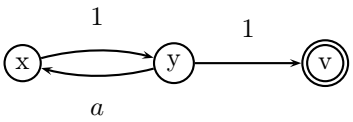
In order to show that I_1 holds after Line 22 we have to show that $\mathsf{dist}(v) = d(v)$ holds (in Line 16 v is added to U, and the values of dist remain unchanged for all nodes in $U \cup \{v\}$). For the sake of contradiction we assume there is a walk $w = vv^2 \dots v^m$ in M with $v^m \in V_T$ and $c(w) \sqsupset \mathsf{dist}(v)$. Due to I_3 this walk has an inner node which is not in $U - \{v\}$, and denote by v^i the last node on this walk not in $U - \{v\}$. Then we have $c(v^i v^{i+} \dots v^m) \sqsupseteq c(w)$ because of cumulativity, $\mathsf{dist}(w) \sqsupseteq c(v^i v^{i+1} \dots v^m)$ and by transitivity of $\sqsupseteq$ we obtain $\mathsf{dist}(w) \sqsupseteq c(w)$. Moreover, by assumption we have $c(w) \sqsupset \mathsf{dist}(v)$, which leads altogether to $\mathsf{dist}(w) \sqsupset \mathsf{dist}(v)$. But this is a contradiction to the choice of v in Line 15.

For I_3 we consider an arbitrary node $u \notin U \cup \{v\}$ and show that after Line 22 the equality $\mathsf{dist}(u) = d(u)$ in $M|_{U \cup \{u,v\}}$ holds. So consider an optimal walk $uu^1u^2 \dots u^k$ with $u^k \in V_T$ in $M_{U \cup \{u\}}$. Then either u^1 equals v or does not equal v. In Line 18 the algorithm compares the costs of walks from u into V_T with inner nodes in $U - \{v\}$ to the costs of walks from u into V_T with v as second node. So the array dist is updated consistently with I_3. Clearly, by construction $(mp(u, \mathsf{succ}))$ produces such an optimal walk from u into V_T, so I_2 is also preserved.

This shows that every finite uniquely labelled target model with cumulative associated s-dioid is refinable. But then every finite (not necessarily uniquely labelled) finitely

labelled target model is refinable due to Corollary 8.3.1: First construct its label-optimised target model and refine it afterwards. The result is an optimal submodel of the original one.

It remains to show that for every not-cumulative dioid there is a target model with it as associated dioid which is not refinable. Therefore we consider the following target model where a is assumed to be an arbitrary element of a non-cumulative dioid:



We show that contrary to the assumption the associated s-dioid has to be cumulative. An optimal submodel can either contain the edge (y, x) or not. If it contains the edge (y, x) we have $1 = c(yv) = c(yxyv) = a$ because every walk leading into v has to be optimal, and hence $a \sqsubseteq 1$ for arbitrary a. Since a was chosen arbitrarily the dioid is cumulative. Assume now there is an optimal submodel which does not contain the edge (y, x). Then in the original (not refined) target model $d(y) = c(yv)$ has to hold (because in the refined model only the walk yv from y to v exists), and the definition of d yields $c(yxyv) \sqsubseteq c(yv)$, which implies $a \sqsubseteq 1$. So in this case too the associated s-dioid has to be cumulative. ■

8.5 Target Models with Non-Cumulative S-Dioids

In analogy to above we consider first only uniquely labelled finite target models without explicit mention. After this we will make the step to a more general class of target models.

The result from Theorem 8.4.1 seems rather discouraging if we want to construct optimal submodels of target models with associated non-cumulative s-dioids. But we will see that there is a possibility to refine such target models if the associated labelled graph does not contain negative cycles. A *negative cycle* is a cycle $x^1x^2 \dots x^n$ with $c(x^1x^2 \dots x^n) \sqsupset 1$. Note that in our setting of general s-dioids multiplication need not be commutative; so $\prod_{i=0}^{n-1} l^i = \prod_{i=0}^{n-1} l^{(i+m) \bmod n}$ for an arbitrary $l \in L(x^1x^2 \dots x^n)$ does not hold in general. To deal with this phenomenon, in [GM08b] the concept of a *pointed cycle* is introduced. However, for our considerations this will not play a role.

If there is no negative cycle then there is always a path p between two nodes x and y with $c(p) = d(x, y)$. Here too this can be shown be removing cycles from any

walk between x and y. Due to the absence of negative cycles this can not decrease the cost (with respect to $\sqsubseteq$) of the walk. So from this follows that under the above assumptions between two reachable nodes there is always a path which is an optimal walk. In particular, between two reachable nodes there is always an optimal walk with edge length at most $|V| - 1$.

In this case we can use an algorithm similar to the Floyd-Warshall algorithm. As a first step we compute by means of Algorithm 9 the distances between every pair of nodes and the successor of every node on an optimal path to every other node.

Algorithm 9 Floyd-Warshall-like Algorithm in the Case of Edge Labels from a General Dioid and a Associated Graph without Negative Cycles

Require: $M = ((\{1..n\}, E), g), a)$ is a finite uniquely labelled target model, $M = ((\{1..n\}, E), g), a)$ is a finite uniquely labelled target model without negative cycles and associated s-dioid $(D, \sum, 0, \cdot, 1)$

```
initialise succ as an n × n-matrix with entries from V
for i ← 1 to n do
    for j ← 1 to n do
        if (i, j) ∈ E then
            dist[i, j] = ℓ(i, j)
        else
            dist[i, j] = 0
        end if
        succ(i, j) = null
    end for
end for
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if dist(i, k) · dist(k, j) ⊐ dist(i, j) then
                dist(i, j) = dist(i, k) · dist(k, j)
                succ(i, j) = succ(i, k)
            end if
        end for
    end for
end for
```

Ensure: $\forall\, i, j \in V$: dist$(i, j) = d(i, j)$; succ encodes optimal walks

In this algorithm succ(i, j) has a similar meaning as in Algorithm 8; the difference is that here no optimal walk from i into the target set V_T is encoded but an optimal

walk from i to j.

The termination of the algorithm is clear. As an invariant for the outermost of the three nested for loops we choose the assertion that $\mathsf{dist}(i,j)$ equals the cost of an optimal walk from i to j whose inner nodes are nodes l with $l < k$, and that succ encodes such a walk.

Before the first entry into the outermost for loop this invariant holds due to the initialisation of succ, dist and k. Now consider the situation after a run through the inner two nested loops. If there is an optimal walk from i to j which does not visit the node k then $\mathsf{dist}(i,k) \cdot \mathsf{dist}(k,j) \sqsupseteq \mathsf{dist}(i,j)$ holds according to Lemma 8.2.1. In the case of equality no update is necessary, so the algorithm has to consider only the case $\mathsf{dist}(i,k) \cdot \mathsf{dist}(k,j) \sqsupseteq \mathsf{dist}(i,j)$. Here the new value of $\mathsf{dist}(i,j)$ is correct, again due to Lemma 8.2.1, and since the newly found walk leads through k the succ-matrix is also updated correctly. Conversely, if there is no optimal walk from i to j with k as an inner node no changes have to be made. The algorithm also leaves dist and succ untouched, since here the condition $\mathsf{dist}(i,k) \cdot \mathsf{dist}(k,j) \sqsupset \mathsf{dist}(i,j)$ becomes false (cf. again Lemma 8.2.1). So the invariant is preserved.

The actual refinement is done in two steps:

1. First, we determine for every node $i \notin V_T$ a target $t(i) \in V_T$ with $d(i) = d(i,t(i))$. This can be done by computing $d(i) = \sum_{j \in V_T} \mathsf{dist}(i,j)$ and storing an appropriate $j \in V_T$.
2. Second, we keep all edges given by $\mathsf{succ}(i,t(i))$ for every $i \notin V_T$.

By construction, the model obtained in this way is an optimal submodel.

8.6 Target Models and Bisimulations

Till now we know some conditions under which a target model is refinable. In this section we investigate the interplay between cost, optimality, refinement and bisimulation equivalences. The first step is the following lemma:

Lemma 8.6.1 *Let $M = (((V,E),g),a)$ be a target model and B a bisimulation equivalence for M. Then for all v in V the target distances of v in M and v/B in M/B coincide.*

Proof: We chose an arbitrary node $v \in V$ and denote the target distance of v in M by $d_M(v)$ and the target distance of v/B in M/B by $d_{M/B}(v/B)$. To proof the

equality $d_M(v) = d_{M/B}(v/B)$ we show the two inequalities $d_M(v) \sqsubseteq d_{M/B}(v/B)$ and $d_{M/B}(v/B) \sqsubseteq d_M(v)$.

So we fix an arbitrary $v \in V$ and consider a walk $w = v^1v^2 \dots v^n$ in M with $v^1 = v$ and $v^n \in V_T$. Let $l \in L(w)$ be an arbitrary labelling of w. By Lemma 6.2.1 we know that M and M/B are bisimilar, so by Lemma 6.1.1 there is walk $w_B = v_B^1 v_B^2 \dots v_B^n$ in M/B with $v^1 \in v_B^1$ and $v_B^n \in (V/B)_T$. Additionally, we can chose this walk in such a way that $l \in (L/B)(w_B)$ holds ($(L/B)(w_B)$ denotes the set of labellings of w_B in M/B). Hence the set of costs of all paths in M leading from v into V_T is a subset of the costs of paths in M/B leading from v/B into $(V/B)_T$. This implies the inequality $d_M(v) \sqsubseteq d_{M/B}(v/B)$.

The reverse inequality can be shown symmetrically (note that we did not use any specific properties of B except that fact that M and M/B are bisimilar and that $\in$ is a bisimulation between M and M/B). ■

This lemma is needed in the proof of the following theorem:

Theorem 8.6.1 *Let $M = (((V, E), g), a)$ be a target model and B a bisimulation equivalence for M such that M/B is finitely labelled. If $(M/B)'_o$ is an optimal target submodel of the label optimised quotient $(M/B)_o$ then $(M/B)'_o \backslash B$ is an optimal target submodel of M.*

Proof: Let $w = v^1v^2 \dots v^n$ be an arbitrary walk in $(M/B)'_o \backslash B$ with $v^n \in V_T$. Due to Theorem 6.3.1 $(M/B)'_o$ and $(M/B)'_o \backslash B$ are bisimilar, so by Lemma 6.1.1 there is the walk $w_B = v^1/Bv^2/B \dots v^n/B$ in $(M/B)'_o$ (note that $\in$ is a bisimulation between $(M/B)' \backslash B$ and $(M/B)'$, and that $\in$ is even a function). Because $(M/B)'_o$ is an optimal target submodel of M/B we have $d_{M/B}(v^1/B) = c_{M/B}(w_B)$ due to Corollary 8.3.1. Lemma 8.6.1 yields $d_M(v^1) = d_{M/B}(v^1/B)$, so we next want to show the equality $c_{M/B}(w_B) = c_M(w)$ which implies that w is an optimal walk in M.

For this purpose consider an arbitrary labelling $\ell^1\ell^2 \dots \ell^{n-1} \in L(w)$ of w in M. By Lemma 6.2.2 we have $\ell^i \in (g/B)(v^i/B, v^{i+1}/B)$ for all $i \in \{1 \dots n-1\}$. Because $(M/B)_o$ is label optimised the inequalities $\ell^i \sqsubseteq (g/B)_o(v^i/B, v^{i+1}/B)$ hold for all $i \in \{1 \dots n-1\}$. But this implies $c_M(w) \sqsubseteq c_{(M/B)_o}(w/B)$ due to isotony of multiplication. As shown in the proof of Lemma 8.2.2 we have $c_{(M/B)_o}(w/B) = c_{(M/B)}(w/B)$, so it remains to show the reverse inequality $c_{(M/B)}(w/B) \sqsubseteq c_M(w)$. Let now $\ell_B^1\ell_B^2 \dots \ell_B^{n-1}$ be the unique labelling of w_B in $(M/B)_o$. Because w is a walk in $(M/B)'_o \backslash B$ we have by construction $\ell_B^i \sqsupseteq \ell^i$ for all $i \in \{1 \dots n-1\}$. Hence the reverse inequality holds by isotony of multiplication.

This shows that every walk in $(M_B)'_o \backslash B$ leading into V_T is optimal, so in order so satisfy Definition 8.3.1 we have to show that in $(M_B)' \backslash B$ the target set is reachable from every node outside of it. But this is an easy consequence of the fact that $(M/B)'_o$

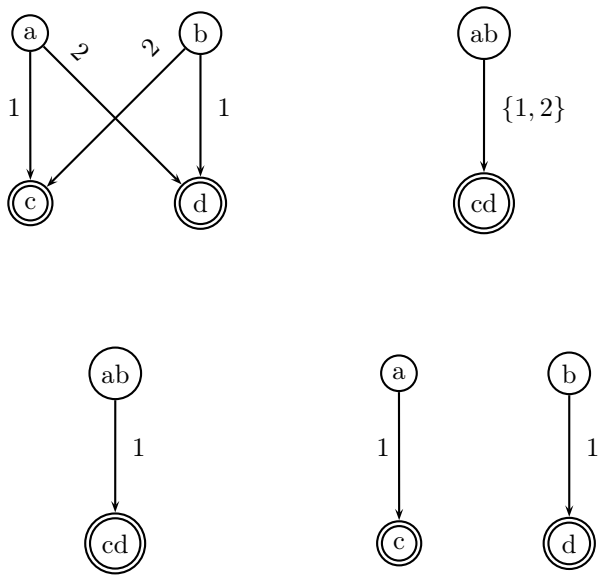


Figure 8.5: A uniquely labelled Target Model (top left) with not uniquely labelled coarsest quotient (top right), its label optimised coarsest quotient (bottom left) and its expansion (bottom right)

is a target model and Lemma 6.1.1 (note that B is left-total!). ■

Remark 1: Clearly the condition that M/B is finitely labelled requires that M is also finitely labelled because of Lemma 6.2.2. □

Remark 2: This Theorem corrects Theorem 4.1. in [Glü11]. The claim made there that the expansion of an optimal submodel of the coarsest quotient is an optimal submodel is refuted by Figure 8.5. If we interpret the labels in the dioid $(\mathbb{R}_0^+ \cup \{\infty\}, \inf, \infty, +, 0)$ (i.e., we consider shortest paths) the top left target model is not optimal because there is the walk ad which is clearly not optimal. Its coarsest quotient in the top right is optimal because the walk $\{ab\}\{cd\}$ has cost 1 despite the fact that the edge $(\{ab\}, \{cd\})$ bears the label $\{1, 2\}$. Its expansion, however, yields the original model which is not optimal. On contrary, the label optimised submodel of the coarsest quotient in the bottom left is also an optimal submodel. Its expansion (in whose construction we have to obey properly Point 2 and 3 of Definition 6.3.1) in the bottom right is indeed an optimal submodel of the initial target model. □

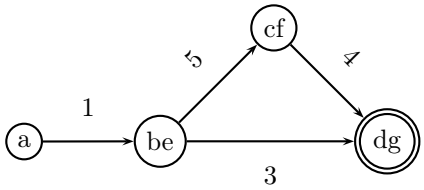


Figure 8.6: A Coarsest Quotient

Theorem 8.6.1 is the central theorem of this section since it gives raise to the generic Algorithm 10.

Algorithm 10 Refining a Target Model via Quotient Construction

Require: $M = (((V, E), g), a)$ is a target model and B is a bisimulation equivalence for M such that M/B is a finite and finitely labelled target model free of negative cycles

1: $(M/B)_o \leftarrow$ label optimised submodel of (M/B)
2: $(M/B)'_o \leftarrow$ optimal target submodel of $(M/B)_o$
3: $M' \leftarrow (M/B)'_o \backslash B$

Ensure: M' is an optimal target submodel of M

The choice of B is arbitrary within the conditions posed on it in the requirement. The refinement in Line 2 is possible by one of the proposed algorithms (Algorithm 8 or Algorithm 9) due to the requirements on M/B (note that the case of a cumulative s-dioid is covered by the absence of negative cycles).

We will illustrate this algorithm a second time (after Remark 2) on the already known target model from Figure 8.1. Its quotient is shown in Figure 8.6. An optimal target submodel of this quotient with respect to the shortest path problem is depicted in Figure 8.7. Eventually, its expansion yields the optimal target submodel from Figure 8.3.

Algorithm 10 is reasonable in two cases (both cases provided that M/B is finitely labelled): First, if M is infinite and M/B is finite, and second, if M is finite and M/B is significantly smaller than M.

In the first case the quotient M/B (and also the computation of the expansion $(M/B)'\backslash B$) has to be done via symbolic execution.

If M is finite one will use for B of course the coarsest bisimulation for M because here the quotient M/B will become smaller as under the use of other possible choices for

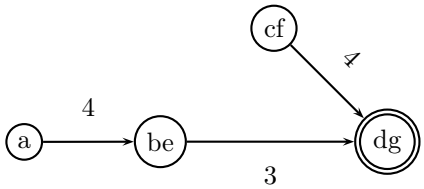


Figure 8.7: An Optimal Quotient Target Submodel

B. Clearly, the expansion in Line 3 can be executed in time $\mathcal{O}(|V|+|E|)$. The coarsest quotient can be determined in $\mathcal{O}(|E| \cdot log(|V|) \cdot |L|)$ time (see Chapter 4), so a speed up (compared to the immediate execution of a refinement algorithm) can be expected only if the runtime of the refinement algorithm in Line (5) is in $\Omega(|E| \cdot log(|V|)$.

If we use Algorithm 8 for the refinement we will hardly achieve an asymptotic speed up. The runtime of this algorithm is in $\mathcal{O}(|V| \cdot log(|V|) + |E|)$ so under the reasonable assumption $|E| \in \Omega(|V|)$ the computation of the coarsest quotient will dominate the execution of the refinement algorithm. However, this consideration abstract of constant factors, so in exceptional cases Algorithm 10 may offer an improvement to the immediate application of Algorithm 8.

However, in the case of the Floyd-Warshall like Algorithm 9 a speed up is possible. To see this we fix an arbitrary finite target model $M = (((V, E), g), a)$ with $V = \{v^1, v^2, \ldots, v^n\}$ and define the model family $(M_m)_{m \in \mathbb{N}^+}$ as follows:

1. $V_m = \bigcup_{1 \leq k \leq m} \{v_m^{ik} \mid v^i \in V\}$,
2. $E_m = \bigcup_{1 \leq k \leq m} \{(v_m^{ik}, v^{jk}) \mid (v^i, v^j) \in E\}$,
3. $g_m(v_m^{ik}, v_m^{jk}) = g(v^i, v^j)$ for all $(v_m^{ik}, v_m^{jk}) \in E_m$ and
4. $a_m(v_m^{ik}) = a(v^i)$ for all $v_m^{ik} \in V_m$.

Intuitively, M_m consists of m copies of M, so we have clearly $|V_m| = m \cdot |V|$ and $|E_m| = m \cdot |E|$ and hence $|E_m| \in \Theta(|V_m|)$. Moreover, the coarsest quotient M_m/B_m of every model M_m is isomorphic to the coarsest quotient M/B of M (cf. the definition from Section 5). The situation is sketched in Figure 8.8.

The computation of the coarsest bisimulation needs $\mathcal{O}(|E_m| \cdot log(|V_m|)) = \mathcal{O}(|V_m| \cdot log(|V_m|))$ time. Line 1 and 2 of Algorithm 10 can be carried out in constant time because the algorithm has to deal with isomorphic models, therefore its execution time is independent of m. Line 3 can be executed in $\mathcal{O}(|V_m| + |E_m|) = \mathcal{O}(V_m)$ time, thus the overall

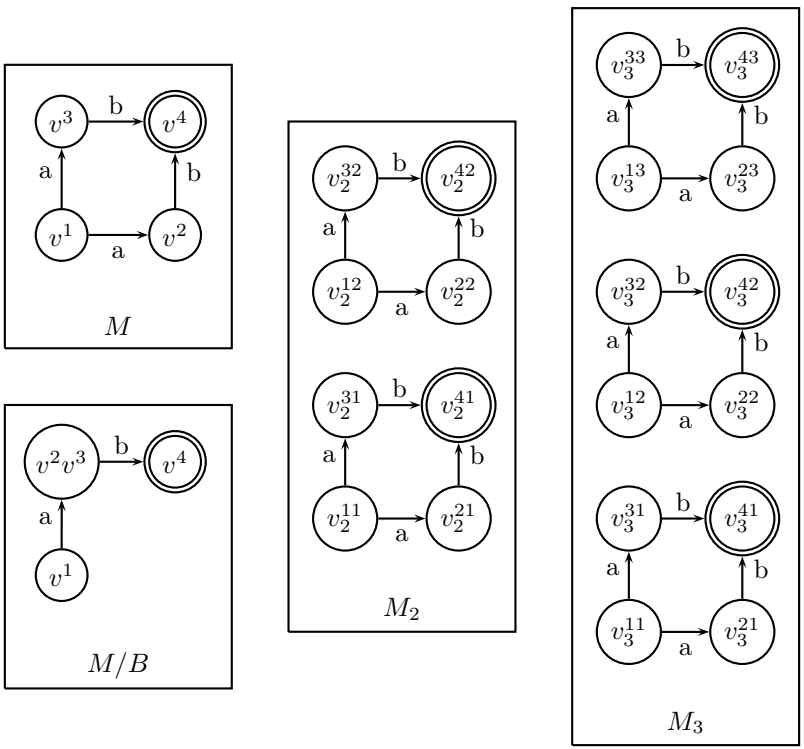


Figure 8.8: A Part of a Model Family M_m with coarsest Quotients isomorphic to M/B

runtime is here in $\mathcal{O}(|V_m| \cdot log(|V_m|))$. In contrast, the immediate application of Algorithm 9 to M_m takes $\mathcal{O}(|V_m|^3)$ time, so Algorithm 10 leads to a better asymptotic runtime as Algorithm 9.

This constructed example does not state anything about real life examples. The speed up will be the higher the the more the size of the original model M is decreased by 'dividing' it by the bisimulation equivalence B. Unfortunately, there is no known technique to estimate the number of equivalence classes of the coarsest bisimulation (which is the key quantity for measuring the shrinking of M compared to M/B) which performs better than the computation of the coarsest bisimulation itself.

There are some heuristic considerations under which circumstances the quotient M/B is smaller than the model M. This holds for models with repeated appearance of identical structures. However, in the case of shortest walks, geographic examples from the real world probably lack this property. Even the road map of highly structured and planned cities as Manhattan, Canberra or Islamabad do not seem to be adequate examples (they have to obey conditions posed by nature like rivers or mountains). An area of application could be problems in chip design. There a lot of repeated and reused structures can be expected by construction. A first impression gives the circuit of the Intel 400 processor from 1971 which can be found on the Intel homepage (see [Intb, Inta]). Unfortunately, most processor circuits are kept secret by the corporations for obvious reasons.

Chapter 9

Linear Fixpoints

In this section we investigate the interplay between bisimulations and linear fixpoint equations. It is a well known fact that optimality problems in graphs can be described using fixpoints functions of the form $f(x) = Ax + b$, where A is a matrix with entries from an s-dioid, and b and x are vectors with analogous entries. This is described e.g. in [GM08b]; Bellman's equations (see e.g. [Jun05]) are of this type. Ironically, in the original paper [Bel58] these equations are called nonlinear. The term nonlinear refers there to classical linear algebra over the real numbers with traditional addition and multiplication. We will see that a certain kind of linear fixpoint equations is compatible with bisimulations.

9.1 Theoretical Considerations

It is a standard approach in graph theory to identify a graph with its adjacency matrix. Because we want to deal also with infinite graphs we will introduce a straightforward generalisation of this:

Definition 9.1.1 Let S be an arbitrary (possibly infinite or even uncountable) set and $(D, \sum, 0, \cdot, 1)$ an s-dioid. An *S-matrix over D* is a mapping $A : S \times S \to D$. A *column S-vector over D* is a mapping $v : S \times \{*\} \to D$, where $\{*\}$ denotes an arbitrary singleton set. Analogously, a *row S-vector over D* is a mapping $v : \{*\} \times S \to D$.

If S is clear from the context it can also be omitted. We also use the term *vector* as an abbreviation for row or column vector. The set of all S-matrices over D is denoted by $D^{S\times S}$, the set of all column S-vectors over D by $D^{S\times *}$, and the set of all row S-vectors over D by $D^{*\times S}$.

As usual, we denote matrices by upper case letters A, B, ... with possible indices and other modifiers. Vectors of both kinds are denoted by lower case letters b, x, v, ..., depending on the usage (b denotes usually a constant vector, x a variable of a function and v an arbitrary vector). Whether a lower case letter denotes a column or a row vector becomes clear from the context. We write briefly A_{st} instead of $A(s,t)$ for a matrix A, and v_s instead of $v(s,*)$ and $v(*,s)$ for a vector. The *sum* v^1+v^2 of two S-column or two S-row vectors v^1 and v^2 over the same s-dioid $(D,\sum,0,\cdot,1)$ is an S-column or S-row vector, resp., over D, defined by $(v^1+v^2)_s =_{df} v^1_s + v^2_s$.

Obviously, an S-vector b over an s-dioid $(D,\sum,0,\cdot,1)$ corresponds to a mapping from S into D. So it induces a partition $\mathcal{S}=\{S_i \mid i\in I\}$ of S, defined by $\forall i,j\in I: i=j \Leftrightarrow \forall s_i\in S_i \forall s_j\in S_j : b_{s_i}=b_{s_j}$. This is called the *partition induced by* b.

With these writing conventions we define the multiplication of a matrix and a vector in a natural way:

Definition 9.1.2 Let A be an S-matrix over an s-dioid $(D,\sum,0,\cdot,1)$, and let v and w be an S-column and an S-row vector resp. over D. Then we define the *product* $A\cdot v$ as an S-column vector over D, given by $(A\cdot v)_s=\sum\limits_{t\in S} A_{st}\cdot v_t$. Analogously, we define the product $w\cdot A$ as an S-row vector over D, given by $(v\cdot A)_s=\sum\limits_{t\in S} v_t\cdot A_{ts}$.

In the adjacency matrix of a graph often the value ∞ is used in order to express the nonexistence of an edge. However, this does not fit our dioid based framework in general. The use of ∞ in the common setting is motivated by the shortest path problem (if there is no walk between two nodes then their distance equals ∞). The reason is that $\sum\emptyset=\infty$ holds in the s-dioid $(\mathbb{R}_0^+\cup\{\infty\},\inf,\infty,+,0)$ (which corresponds to the shortest walk problem). This gives rise to the following definition:

Definition 9.1.3 Let $M=(((V,E),g),a)$ be a uniquely labelled model over an s-dioid $(D,\sum,0,\cdot,1)$. Then its *adjacency matrix* A is defined by $A_{v_1v_2}=g(v_1,v_2)$ if $(v_1,v_2)\in E$ and $A_{v_1v_2}=0$ otherwise (recall the convention that for uniquely labelled models g may deliver elements of L instead of subsets). The adjacency matrix of a uniquely labelled graph $G=((V,E),g)$ is defined analogously.

The quotient of a uniquely labelled model or graph need not be uniquely labelled itself (cf. Figure 6.2), so the concept of the adjacency matrix is not directly compatible with bisimulations. To cover this problem for a uniquely labelled model $M=(((V,E),g),a)$

with adjacency matrix A and a bisimulation B for M we define the *quotient matrix* A/B by $(A/B)_{v/Bw/B} = \sum\{A_{v'w'} \mid v' \in v/B, w' \in w/B\}$. For this construction we have the following lemma:

Lemma 9.1.1 *Let $M = (((V,E),g),a)$ be a uniquely labelled model over an s-dioid $(D, \sum, 0, \cdot, 1)$, A its adjacency matrix and B be a bisimulation equivalence for M. Then for every $(v,w) \in E$ the equality $\sum_{w' \in w/B} A_{vw'} = (A/B)_{(v/B)(w/B)}$ holds.*

Proof: It suffices to show the equality $\bigcup_{w' \in w/B} g(v,w') = g/B(v/B, w/B)$ for arbitrary $(v,w) \in E$. First, we pick an arbitrary $\ell \in \bigcup_{w' \in w/B} g(v,w')$. Then there is a $w' \in w/B$ with $\ell = g(v,w')$ (recall that M is uniquely labelled). This implies $\ell \in g/B(v/B, w/B)$ by definition of g/B (cf. Definition 6.2.1). Conversely, let $\ell \in g/B(v/B, w/B)$ be arbitrarily chosen. Then there are $v' \in v/B$ and $w'' \in w/B$ with $g(v', w') = \ell$. Because B is an equivalence we also have $v \in v/B$ and hence by definition $v'Bv$. But by definition of a bisimulation there has to be a w' with $w''Bw'$ such that $v \xrightarrow{\ell}_g w'$ holds, which implies $\ell \in \bigcup_{w' \in w/B} g(v,w')$. ■

After we established a connection between adjacency matrices and bisimulations we will do the same for vectors and bisimulations. So given a model $M = (((V,E),g),a)$, a bisimulation B for M and a row (V/B)-vector we define the *expansion* $x_B\backslash B$ of x_B as a row V-vector, given by $(x_B\backslash B)(*,v) =_{df} x_B(*, v/B)$. The expansion of a row S-vector is defined analogously.

Theorem 9.1.1 *Let $M = (((V,E),g),a)$ and A be as in Lemma 9.1.1, let b be a V-column vector over D and B be a bisimulation equivalence for M respecting the partition induced by b. Let x_B be an (V/B)-column vector over D such that $x_B = (A/B)x_B + b/B$ holds. Then also $x_B\backslash B = A(x_B\backslash B) + b$ holds.*

Proof: First we argue as follows:

$$
\begin{array}{l}
x_B\backslash B = A(x_B\backslash B) + b \Leftrightarrow \\
\quad \{ \text{ pointwise equality } \} \\
\forall v \in V : (x_B\backslash B)_v = (A(x_B\backslash B))_v + b_v \Leftrightarrow \\
\quad \{ \text{ definition of matrix product } \} \\
\forall v \in V : (x_B\backslash B)_v = \sum_{w \in V} A_{vw} \cdot (x_b\backslash B)_w + b_v
\end{array}
$$

To get rid of the annoying universal quantifier we now fix an arbitrary $v \in V$ and argue further:

$$
\begin{array}{l}
(x_B\backslash B)_v = \sum\limits_{w\in V} A_{vw}\cdot(x_b\backslash B)_w + b_v \;\Leftrightarrow\\
\quad \{\text{ definition of expansion of a vector }\}\\
(x_B)_{v/B} = \sum\limits_{w\in V} A_{vw}\cdot(x_b\backslash B)_w + b_v \;\Leftrightarrow\\
\quad \{\text{ assumption } x_B = (A/B)x_B + b/B\text{, pointwise equality,}\\
\quad\ \text{definition of matrix product }\}\\
\quad \sum\limits_{w/B\in V/B} (A/B)_{v/Bw/B}\cdot(x_B)_{w/B} + (b/B)_{v/B} = \sum\limits_{w\in V} A_{vw}\cdot(x_b\backslash B)_w + b_v
\end{array}
$$

Because B is a bisimulation respecting the partition induced by b we have $(b/B)_{v/B} = b_v$, so it suffices to show the equality $\sum\limits_{w/B\in V/B} (A/B)_{v/Bw/B}\cdot(x_B)_{w/B} = \sum\limits_{w\in V} A_{vw}\cdot(x_b\backslash B)_w$. This can be shown as follows:

$$
\begin{array}{l}
\sum\limits_{w\in V} A_{vw}\cdot(x_b\backslash B)_w =\\
\quad \{\text{ definition of expansion of a vector }\}\\
\sum\limits_{w\in V} A_{vw}\cdot(x_b)_{w/B} =\\
\quad \{\text{ sum splitting }\}\\
\quad \sum\limits_{w/B\in V/B}\sum\limits_{w\in w/B} A_{vw}\cdot(x_b)_{w/B} =\\
\quad \{\text{ Lemma 9.1.1, distributivity }\}\\
\quad \sum\limits_{w/B\in V/B} (A/B)_{v/Bw/B}\cdot(x_B)_{w/B},
\end{array}
$$

and we are done. ■

By symmetric argumentation we obtain the following corollary:

Corollary 9.1.1 *Let $M = (((V,E),g),a)$ be a uniquely labelled model over an s-dioid $(D,\sum,0,\cdot,1)$, A its adjacency matrix, b a V-row vector over D and B be a bisimulation equivalence for M respecting the partition induced by b. Let x_B be a (V/B)-row vector over D such that $x_B = x_B(A/B) + b/B$ holds. Then also $x_B\backslash B = (x_B\backslash B)A + b$ holds.*

Theorem 9.1.1 and Corollary 9.1.1 show under which conditions linear fixpoint equations are compatible with bisimulations. In general, such equations of the form $x = Ax + b$ or $x = xA + b$ have more than one solution in x. For example, consider the matrix $1^{V\times V}$ given by $1^{V\times V}_{vw} = 0$ for all $v,w\in V$ with $v\neq w$ and $1^{V\times V}_{vv} = 1$ for all $v\in V$, and the column vector 0^V, given by $0^V_v = 0$ for all $v\in V$. Then every V-column vector is a solution of $x = 1^{V\times V}x + 0^v$. However, if we introduce an order $\leq_V$ on all V-vectors over $(D,\sum,0,\cdot,1)$, given by $v^1\leq_V v^2 \Leftrightarrow_{df} \forall x\in V : v^1_x \leq v^2_x$, we have exactly one least solution with respect to $\leq_V$. This is due to the fact that $D^{V\times *}$ and $D^{*\times V}$ ordered by $\leq_V$ form complete lattices, and that the functions $f(x) = Ax+b$

and $g(x) = xA + b$ are isotone with respect to $\leq_V$. So the least fixpoints ν_f and ν_g of f and g can be written as $\nu_f = \sum_{i\geq 0} f^i(0_V)$ and $\nu_g = \sum_{i\geq 0} g^i(0_V)$, where $\sum$ denotes the supremum with respect to $\leq_V$, f^i and g^i denote the i-fold application of f and g, and 0_V is the V-vector, given by $(0_V)_v = 0$ for all $v \in V$.

This is due to the famous fixpoint iteration theorem of Kleene, named after [Kle52]. There it occurs in the context of partial recursive functions in computability theory. A more modern formulation in the framework of lattices was given a few years later in [Tar55]. However, the intellectual authorship remains open.

By means of Theorem 9.1.1 and Corollary 9.1.1 we know how to compute a fixpoint of a linear equation from a fixpoint of a quotient. The next theorem gives a result connecting the least fixpoints of an equation and its quotient (where the term 'least' has to be read in the sense of the above explanations):

Theorem 9.1.2 *Let $M = (((V, E), g), a)$, $(D, \sum, 0, \cdot, 1)$, A, b and B as in Theorem 9.1.1. Let y be the least fixpoint of the linear function $f(x) = Ax + b$, and denote by y_B be the least fixpoint of the function $f_B(x_B) = (A/B)x_B + b/B$. Then the equality $y = y_B \backslash B$ holds.*

Proof: As already mentioned above, we have the equations $\nu_f = \sum_{i\geq 0} f^i(0_V)$ and $\nu_{f_B} = \sum_{i\geq 0} f_B^i(0_{V/B})$ for the least fixpoints ν_f and ν_{f_B} of f and f_B, resp.

We will show that for all $i \in \mathbb{N}$ the equality $f_B^i(0_{V/B})\backslash B = f^i(0_V)$ holds. This implies the claim because both $\leq_V$ and $\leq_{V/B}$ are product orders, so the suprema with respect to these orders are the pointwise suprema with respect to the dioid order $\leq$. This equality can be shown by simple induction:

Induction base: Here we have to show $0_{V/B}\backslash B = 0_V$ which is trivially true due to the definition of the expansion.

Induction step: We fix an arbitrary $v \in V$ and assume $f_B^i(0_{V/B})\backslash B = f^i(0_V)$ for an arbitrary fixed $i \in \mathbb{N}$. Then we have to show the equality $A(f^i(0_V))_v + b_v = (A/B)(f_B^i(0_{V/B}))_{v/B} + (b_B)_{v/B}$, according to the definition of the expansion operation $\backslash B$. Because B respects the partition induced by b the equality $b_v = (b_B)_{v/B}$ holds trivially. So it suffices to show the equality $A(f^i(0_V))_v = (A/B)(f_B^i(0_{V/B}))_{v/B}$. This can be done by the following calculation:

$$
\begin{aligned}
&A(f^i(0_V))_v = \\
&\quad \{ \text{ definition of matrix multiplication } \} \\
&\sum_{w\in V} A_{vw}(f^i(0_V))_w = \\
&\quad \{ \text{ sum splitting } \}
\end{aligned}
$$

$$\sum_{w/B \in V/B} \sum_{w \in w/B} A_{vw}(f^i(0_V))_w =$$
{ induction hypothesis, definition of $\backslash B$ }
$$\sum_{w/B \in V/B} \sum_{w \in w/B} A_{vw}(f_B^i(0_{V/B}))_{w/B} =$$
{ Lemma 9.1.1, distributivity }
$$\sum_{w/B \in V/B} (A/B)_{v/B\,w/B}(f_B^i(0_{V/B}))_{w/B} =$$
{ definition of multiplication }
$$(A/B)(f_B^i(0_{V/B}))_{v/B}$$

So the proof is finished. ■

Intuitively this proof executes Kleene's fixpoint iteration for ν_f and ν_{f_B} in parallel on a model and a quotient of it. In every step the values assigned to a node v and its associated quotient node v/B coincide, so the sequences have the same supremum. Note also the similarity to the proof of Theorem 7.2.1.

As above, we obtain the following corollary by a symmetric argumentation:

Corollary 9.1.2 *Let $M = (((V, E), g), a)$ be a uniquely labelled model over an s-dioid $(D, \sum, 0, \cdot, 1)$, A its adjacency matrix, b a V-row vector over D and B be a bisimulation equivalence for M respecting the partition induced by b. Let y be the least fixpoint of the linear function $f(x) = xA + b$, and let y_B be the least fixpoint of the function $f_B(x_B) = x_B(A/B) + b/B$. Then the equality $y = y_B\backslash B$ holds.*

9.2 Interpretations

The fixpoints of linear equations are closely connected with target models. Consider a uniquely labelled target model $M = (((V, E), g), a)$ over an s-dioid $(D, \sum, 0, \cdot, 1)$, let A be its adjacency matrix and let b be the V-row vector over D, defined by $b_v = 0$ for all $v \in V$ with $a(v) = \mathsf{trans}$, and $b_v = 1$ for all $v \in V$ with $a(v) = \mathsf{fin}$. In this setting, the target distance $d(x)$ can also be viewed as a V-row vector over D. Moreover, this vector is the least solution in x of the linear equation $x = xA + b$ (see e.g. [BHZ77, Car71, GM08b]). This fact and Corollary 9.1.2 give an alternative proof of Theorem 8.6.1.

The question arises whether a linear equation $x = xA+b$ has an interpretation similar to the already mentioned one as in [BHZ77, Car71, GM08b] if b takes other values except 0 and 1. Such kind of b's could be used to model some kinds of reward, penalty or import at a given node.

Another nearby question is whether the ideas from this chapter can be carried over to classical linear algebra. Unfortunately, with high probability the answer is 'No'. The

algebraic properties of multiplication and addition in both contexts differ too much. The highest obstacle will be the idempotency of addition which plays an important role in all proofs here.

Chapter 10

Stochastic Games

This chapter deals with stochastic games, a problem for which no provably polynomial algorithm is known. The only known fact about its complexity is that it is in $\mathcal{NP} \cap \mathsf{co}\mathcal{NP}$. We first give an introduction to stochastic games and their properties, and investigate afterwards its compatibility with the quotient construction.

10.1 An Introduction to Stochastic Games

The origin of stochastic games lies in the paper [Sha53]. There two players make, in every position from a finite position set, independently a choice between a finite number of alternatives. With a certain probability, depending on the position and the choices made by the players, the game stops, or the game is continued in other positions with associated probabilities. We will consider a similar model, which was also investigated in [Con] and [Con92]. The definition there is translated into our model formalism as follows:

Definition 10.1.1 A *simple stochastic game (SSG)* is a model $M = (((V, E), g), a)$ with $g(e) = *$ for all $e \in E$ and $a : V \to \{\mathsf{max}, \mathsf{min}, \mathsf{average}, 0, 1\}$. For a, the restrictions $\{v \in V \mid a(v) = 0\} = \{v \in V \mid a(v) = 1\} = 1$ and $\{v \in V \mid a(v) = 0\} \cap \{v \in V \mid a(v) = 1\} = \emptyset$ hold. Moreover, every node v with $a(v) \in \{\mathsf{max}, \mathsf{min}, \mathsf{average}\}$ has an outdegree in $\{1, 2\}$, and every node v with $a(v) \in \{0, 1\}$ has no outgoing edge.

We abbreviate the sets $\{v \in V \mid a(v) = \mathsf{max}\}$, $\{v \in V \mid a(v) = \mathsf{min}\}$ and $\{v \in V \mid a(v) = \mathsf{average}\}$ by V_{max}, V_{min} and V_{average}, resp., and call their elements max , min and average nodes. The nodes v with $a(v) = 0$ and $a(v) = 1$ are called the *0-sink* and *1-sink*, resp., are denoted by $sink_0$ and $sink_1$ and are referred to as the *sink nodes* of M. It will become clear that the edge labels are here not of interest; so we label them with an arbitrary value. In contrast to the definition in [Con] and [Con92] we allow outdegrees in $\{1, 2\}$ for no-sink nodes (there these outdegrees have to be exactly 2). The reason will become clear when we consider quotients of simple stochastic games.

The graph of an SSG is used as a game base for two players, a min-player and a max-player. Every player chooses a strategy, corresponding to a subset of E, before the game starts. The min-player chooses a subset $\tau \subseteq (V_{\mathsf{min}} \times V) \cap E$ such that every node in V_{min} has exactly one outgoing edge in τ. Symmetrically, the max-player chooses a subset $\sigma \subseteq (V_{\mathsf{max}} \times V) \cap E$ such that every node in V_{max} has exactly one outgoing edge in σ. The strategy of the min-player is called min-strategy, of the max-player max-strategy.

A pair (σ, τ) for a SSG $M = (((V, E), g), a)$ induces a model $M_{\sigma,\tau} = (((V_{\sigma,\tau}, E_{\sigma,\tau}), g_{\sigma,\tau}), a_{\sigma,\tau})$, given by $V_{\sigma,\tau} = V$, $E_{\sigma,\tau} = \sigma \dot\cup \tau \dot\cup (V_{\mathsf{average}} \times V) \cap E$. Intuitively, $M_{\sigma,\tau}$ arises from M by removing the edges which were discarded by the players when choosing their strategies σ and τ. A special requirement we will impose from now on for every SSG M is that for every pair (σ, τ) of max- and min-strategies there is a walk in $M_{\sigma,\tau}$ from an arbitrary node into a sink node. This property is also called *stopping*. In this case, for every pair of max- and min-strategies (σ, τ) a random walk in $M_{\sigma,\tau}$ reaches a sink node with probability 1.

After the players made their strategy choices σ and τ, a token is initially placed on an arbitrary node $v \in V$ and moved along the edges of $M_{\sigma,\tau}$ till it reaches a sink node. On a max or min node the token is moved along the only outgoing edge. If the token is on an average node with one outgoing edge it is moved along this edge. On an average node with two outgoing edges one of the edges is chosen randomly with probability $\frac{1}{2}$ each and the token is moved along this edge. If the token reaches the 0-sink the min-player wins, and the max-player wins if it reaches the 1-sink.

For a node v and a pair of strategies (σ, τ) we define the *value* $val_{\sigma,\tau}$ of v with respect to (σ, τ) as the probability that the max-player wins if the strategies σ and τ were chosen and the token was initially placed on v. Given a SSG and a pair of strategies the values with respect to the given strategies can be computed in polynomial time. The *optimal value* $val(v)$ of a node v is then defined by $val(v) = \max_{\sigma} \min_{\tau} val_{\sigma,\tau}$. This definition may seem asymmetric due to the order of the max- and min-operators, but Shapley shows in [Sha53] the equality $\max_{\sigma} \min_{\tau} val_{\sigma,\tau} = \min_{\tau} \max_{\sigma} val_{\sigma,\tau}$. A max-strategy σ is called *optimal* if $val(v) = \min_{\tau'} val_{\sigma,\tau'}$ holds. An optimal min-strategy is

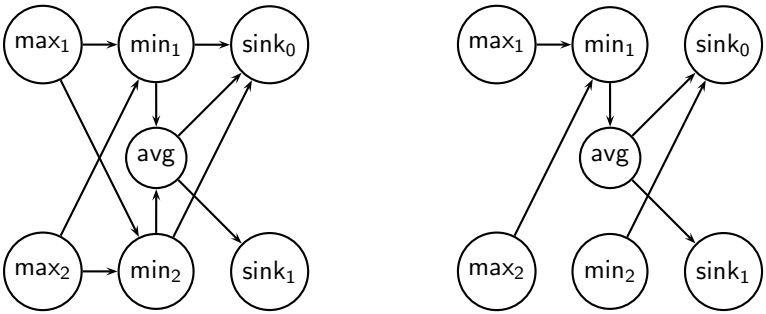


Figure 10.1: An SSG (left) and the result after choosing a pair of strategies (right)

defined symmetrically.

The *SSG problem* as a decision problem is to decide whether the optimal value of a given node is greater than or equal to $\frac{1}{2}$. This problem is in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$, as shown first in [Con92]. Moreover, there is no known algorithm for this problem running in polynomial time. In contrast, given a max-strategy σ the computation of a min-strategy τ satisfying for all nodes v the equality $val_{\sigma,\tau}(v) = \min_{\tau'} val_{\sigma,\tau'}(v)$ can be done in polynomial time by means of linear programming (the associated linear program is constructed in [Der72], and its solvability in polynomial time is shown in [Kya66]). Such a min-strategy is called *optimal with respect to* σ.

An example for an SSG is given in the left part of Figure 10.1. The nodes are captioned in a natural way, i.e. the set of max nodes is $\{\mathsf{max}_1, \mathsf{max}_2\}$, the min nodes are $\{\mathsf{min}_1, \mathsf{min}_2\}$, there is only one average node, namely avg, and the 0- and 1-sink are sink_0 and sink_1.

The right part of Figure 10.1 shows the situation after the players chose the pair of strategies $(\sigma, \tau) = (\{(\mathsf{max}_1, \mathsf{min}_1), (\mathsf{max}_2, \mathsf{min}_1)\}, \{(\mathsf{min}_1, \mathsf{avg}), (\mathsf{min}_2, \mathsf{sink}_0)\})$. Clearly, we have $val_{\sigma,\tau}(\mathsf{sink}_0) = 0$ and $val_{\sigma,\tau}(\mathsf{sink}_1) = 1$. Hence we have also $val_{\sigma,\tau}(\mathsf{min}_2) = 0$. Since at the node avg both edges are chosen with probability $\frac{1}{2}$ each, we have $val_{\sigma,\tau}(\mathsf{avg}) = \frac{1}{2}$. Therefore we obtain $val_{\sigma,\tau}(\mathsf{min}_1) = val_{\sigma,\tau}(\mathsf{max}_1) = val_{\sigma,\tau}(\mathsf{max}_2) = \frac{1}{2}$. A general method for determining the values with respect to a given pair of strategies can be derived from the theory of Markov processes.

However, it is obvious that the min-player can obtain a better strategy τ' by choosing $(\mathsf{min}_1, \mathsf{sink}_0)$ instead of $(\mathsf{min}_1, \mathsf{avg})$. Then the values of min_1, max_1 and max_2 with respect to the new strategy pair (σ, τ') become zero instead of $\frac{1}{2}$. It is also easy to see that τ' is an optimal min-strategy, as well as σ is an optimal max-strategy

(unfortunately for him, the max-player can not do better). So we have $val(\mathsf{min}_1) = val(\mathsf{min}_2) = val(\mathsf{max}_1) = val(\mathsf{max}_2) = val(\mathsf{sink}_0) = 0$, $val(\mathsf{avg}) = \frac{1}{2}$ and $val(\mathsf{sink}_1) = 1$.

An important fact for our further considerations is that the optimal values are the unique solutions of the following system of equations:

$$
\begin{array}{ll}
val(i) = val(j) & \text{if } i \text{ has outdegree 1} \\
val(i) = \max\{val(j), val(k)\} & \text{if } i \text{ is a max node with children } j \text{ and } k \\
val(i) = \min\{val(j), val(k)\} & \text{if } i \text{ is a min node with children } j \text{ and } k \\
val(i) = \frac{1}{2}(val(j) + val(k)) & \text{if } i \text{ is an average node with children } j \text{ and } k \\
val(i) = 0 & \text{if } i \text{ is a 0-sink} \\
val(i) = 1 & \text{if } i \text{ is a 1-sink}
\end{array}
$$

We call these equations *optimality equations*.

Remark: In [Con] all nodes except the sink nodes are restricted to have exactly two outgoing edges. Consequently, the first optimality equation is not given there. However, there is an easy construction which shows that for every SSG according to Definition 10.1.1 there is an 'equivalent' SSG according to [Con]. Therefore take a look at the left part of Figure 10.2. Given an SSG M in our sense, we introduce for each node v^1 with exactly one outgoing edge to v^2 a new average node avg and add the edges shown in Figure 10.2. This construction yields an SSG M' in the sense of [Con]. Once the token reaches v^1 it will in both SSG's reach the node v^2 with probability 1 either directly or after some visits of avg. Hence the optimal values of v^1 in M and M' coincide. Furthermore, we can apply now the equations given in [Con] (corresponding to the last five optimality equations in our sense) to v^1 in M'.

If v^1 is an average node this yields the system

$$
\begin{array}{l}
val(v^1) = \frac{1}{2}(val(v^1) + val(avg)) \\
val(avg) = \frac{1}{2}(val(v^1) + val(v^2))
\end{array}
$$

Elementary algebra leads to $val(v^1) = val(v^2)$. If v^1 is a min node we get the system

$$
\begin{array}{l}
val(v^1) = \min\{val(v^2), val(avg)\} \\
val(avg) = \frac{1}{2}(val(v^1) + val(v^2))
\end{array}
$$

Assume first that $val(v^1) < val(v^2)$. Then we have $val(v^1) < val(avg) < val(v^2)$. This implies $val(v^1) = \min\{val(v^2), val(avg)\} = val(v^2)$ which contradicts the assumption $val(v^1) < val(v^2)$. Symmetrically the case $val(v^1) > val(v^2)$ can be ruled out, so $val(v^1) = val(v^2)$ has to hold. In an analogous manner we can obtain

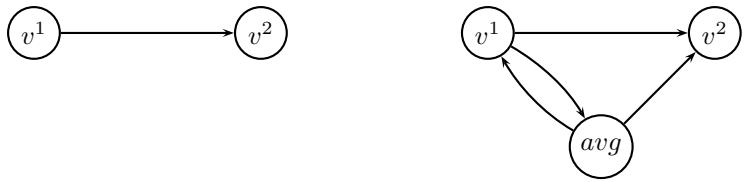


Figure 10.2: Replacing of a single outgoing edge

$val(v^1) = val(v^2)$ if v^1 is a max node, which justifies our first optimality equation. □

10.2 Stochastic Games and Bisimulation Quotients

Because SSG's are defined as models we can apply the tools from Section 6. In particular, we could try to construct a quotient of an SSG. However, we have to ensure that due to the constraints posed in Definition 10.1.1 on the nodes' degrees this operation is well-defined and yields again an SSG. For this purpose we need the following lemma:

Lemma 10.2.1 *Let $M = (((V, E), g), a)$ be a model and B a bisimulation equivalence for M. Then for every node $v \in V$ the following facts about its outdegree $d_{out}(v)$ hold:*

a) $d_{out}(v) \geq d_{out}(v/B)$

b) $d_{out}(v) > 0 \Rightarrow d_{out}(v/B) > 0$

The outdegree $d_{out}(v/B)$ has to be understood in M/B.

Proof: We fix an arbitrary $v \in V$ and argue as follows:

a): Let (w^1/B), (w^2/B), ..., (w^n/B) with $n = d_{out}(v/B)$ be the successors of (v/B) in M/B. Due to Lemmata 6.2.1 and 6.1.1 there have to be nodes $\hat{w}^1$, $\hat{w}^2$, ..., $\hat{w}^n$ in V with $\hat{w}^i \in (w^i/B)$ and $(v, \hat{w}^i) \in E$ for all $i \in \{1 \dots n\}$. Because B is an equivalence the sets (w^1/B), (w^2/B), ..., (w^n/B) are disjoint and hence the nodes $\hat{w}^1$, $\hat{w}^2$, ..., $\hat{w}^n$ are distinct, which implies the claim.
b): Let $\hat{w}$ be an arbitrary successor of v. Again due to Lemmata 6.2.1 and 6.1.1 there is a node (w/B) such that (w/B) is a successor of (v/B) in M/B (more precisely, $(v/B, \hat{w}/B) \in E/B$ holds). ■

This lemma shows that the quotient of an SSG is again an SSG. The left part of Figure 10.3 show the coarsest quotient of the SSG from Figure 10.1 with self-explanatory

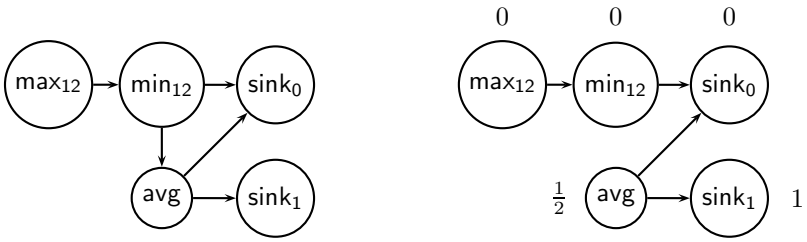


Figure 10.3: A quotient SSG (left) and the result after choosing a pair of optimal strategies (right), together with optimal node values

node captions. In this quotient, the node max_{12} has an outdegree of 1. This phenomenon was the reason for the difference in the definition of an SSG between [Con92] and Definition 10.1.1.

Our goal is to construct a pair of optimal strategies, or equivalently, the node values for an SSG via its quotient. We will have to be cautious since the straightforward expansion of an optimal strategy in general does not yield a valid strategy. So the expansion of the max-strategy from the right part of Figure 10.3 in the spirit of Definition 6.3.1 would yield the set of edges $\{(\mathsf{max}_1, \mathsf{min}_1), (\mathsf{max}_1, \mathsf{min}_2), (\mathsf{max}_2, \mathsf{min}_1), (\mathsf{max}_2, \mathsf{min}_2)\}$, which is no max-strategy by definition. However, we will see that the node values are compatible with the quotient and expansion operation. This is the content of the following theorem:

Theorem 10.2.1 *Let $M = (((V, E), g), a)$ be an SSG and B a bisimulation equivalence for M, denote by val_B the optimal value function in M/B and by val the optimal value function in M. Define the function $val_B \backslash B : v \to [0, 1]$ by $(val_B \backslash B)(v) = val_B(v/B)$. Then for every node $v \in V$ the equality $val(v) = (val_B \backslash B)(v)$ holds.*

Proof: We fix an arbitrary $v \in V$ and show that $val_B \backslash B$ fulfills the optimality equations. Then val and $val_B \backslash B$ coincide due to the uniqueness of the optimal value function. To this purpose we make the following case distinctions:

a) v/B is an average node with outdegree 1 and v has outdegree 2.

b) v/B is an average node with outdegree 1 and v has outdegree 1.

c) v/B is an average node with outdegree 2.

d) v/B is a max (min) node with outdegree 1 and v has outdegree 2.

e) v/B is a max (min) node with outdegree 1 and v has outdegree 1.

f) v/B is a max (min) node with outdegree 2.

g) v/B is a sink node.

Case a): Let w/B be the successor of v/B in M/B, and let w_1 and w_2 be the successors of v in M. By Lemmata 6.2.1 and 6.1.1 we have $w_1/B = w_2/B = w/B$. Because val_B is an optimal value function the equality $val_B(v/B) = val_B(w/B)$ holds, so by construction we obtain $(val_B\backslash B)(v) = (val_B\backslash B)(w_1) = (val_B\backslash B)(w_2)$. Hence the condition $(val_B\backslash B)(v) = \frac{1}{2}\ ((val_B\backslash B)(w_1) + (val_B\backslash B)(w_2))$ is fulfilled.

Case b): Let w/B be the successor of v/B in M/B. Then there is a $w' \in w/B$ which is the only successor of v in M. Because val_B fulfills the optimality equations we have $val_B(v/B) = val_B(w/B)$. So by construction of $val_B\backslash B$ we have in accord with the optimality equations also $(val_B\backslash B)(v) = (val_B\backslash B)(w')$.

Case c): Let w_1/B and w_2/B be the successors of v/B in M/B. Due to Lemma 10.2.1 v has an outdegree of at least 2, and because M is an SSG v has exactly two successors w_1' and w_2' in M with $w_1' \in w_1/B$ and $w_2' \in w_2/B$. val_B is an optimal value function, so $val_B(v/B) = \frac{1}{2}\cdot\ (val_B(w_1/B) + val_B(w_2/B))$ holds. But then by construction of $val_B\backslash B$ we have $(val_B\backslash B)(v) = \frac{1}{2}\cdot\ ((val_B\backslash B)(w_1') + (val_B\backslash B)(w_2'))$ which fulfills the optimality equations.

For the Cases d) - f) we assume that v is a max node; min nodes can be treated symmetrically.

Case d): Let w/B be the successor of v/B in M/B, and let w_1 and w_2 be the successors of v in M. Analogously to Case a) we have $w_1/B = w_2/B = w/B$ and $val_B(v/B) = val_B(w/B)$. By definition of $val_B\backslash B$ we get $(val_B\backslash B)(v) = (val_B\backslash B)(w_1) = (val_B\backslash B)(w_2) = \max\{(val_B\backslash B)(w_1), (val_B\backslash B)(w_2)\}$, so the optimality equations are satisfied.

Case e): As in Case b) let w/B be the successor of v/B in M/B and $w' \in w/B$ the only successor of v in M. Because of $val_B(v/B) = val_B(w/B)$ (which follows from the optimality equations for val_B) we have by construction $(val_B\backslash B)(v) = (val_B\backslash B)(w')$, which fulfills the optimality equations.

Case f): In analogy to Case c) let w_1/B and w_2/B be the successors of v/B in M/B and w_1', w_2' in M with $w_1' \in w_1/B$ and $w_2' \in w_2/B$ the two successors of v in M. Now $val_B(v/B) = \min\{val_B(w_1/B), val_B(w_2/B)\}$ holds, which implies $(val_B\backslash B)(v) = \min\{(val_B\backslash B)(w_1'), (val_B\backslash B)(w_2')\}$, fulfilling the optimality equations.

Case g): Here $(val_B\backslash B)(v)$ obviously fulfills the optimality equations because val_B is an optimal value function and due to the definition of $val_B\backslash B$. ■

This theorem shows the correctness of Algorithm 11. We do not know yet how to compute the optimal value function in Line 2; this will be discussed in Section 10.3.

Algorithm 11 Computing the Optimal Value Function via the Coarsest Quotient

Require: an SSG $M = (((V, E), g), a)$
1: compute the coarsest bisimulation B for P
2: compute the optimal value function val_B of M/B
3: $val \leftarrow val_B \backslash B$
Ensure: val is the optimal value function of M

The idea of the algorithm is illustrated in Figure 10.3: In the right part, the optimal node values, i.e. the values of val_B, are written next to the nodes. The function $val_B \backslash B$ yields the optimal nodes values which we discussed considering the SSG from Figure 10.1.

10.3 Algorithms for Stochastic Games

Before we can investigate the efficiency of Algorithm 11 we have to take a look at known algorithms for SSGs. The algorithms under consideration are taken from [Con]; related approaches can be found e.g. in [Fed80, FSTV91, vdW76, vdW76] and various other sources.

First we will introduce some notation about node labellings we need for the Algorithm in Subsection 10.3.1. Consider a labelling function $\overline{val} : V \to [0, 1]$ on the node set V of an SSG $M = (((V, E), g), a)$ with $\overline{val}(sink_0) = 0$ and $\overline{val}(sink_1) = 1$. Such a function is called *feasible* if for all $i \in V - \{sink_0, sink_1\}$ the following holds:

- if i is an average node with one child j then $\overline{val}(i) = \overline{val}(j)$
- if i is an average node with two children j and k then $\overline{val}(i) = \frac{1}{2}(\overline{val}(j) + \overline{val}(k))$
- if i is a max node with one child j then $\overline{val}(i) \geq \overline{val}(j)$
- if i is a max node with two children j and k then $\overline{val}(i) \geq \max\{\overline{val}(j) + \overline{val}(k)\}$
- if i is a min node with one child j then $\overline{val}(i) \leq \overline{val}(j)$
- if i is a min node with two children j and k then $\overline{val}(i) \leq \min\{\overline{val}(j) + \overline{val}(k)\}$

Moreover, $\overline{val}$ is called *stable* if $\overline{val}$ is feasible and the above inequalities are strengthened to equalities. Clearly, if $\overline{val}$ is stable it is the unique solution of the optimality equations and hence labels the nodes with their optimal values.

For the second algorithm which we will investigate in Subsection 10.3.2 we introduce the idea of a *switchable node*. Given a pair (σ, τ) of max- and min-strategies, we say that a max-node i with two children j and k is (σ,τ)-*switchable* if $val_{\sigma,\tau}(i) < \mathsf{max}\{val_{\sigma,\tau}(j), val_{\sigma,\tau}(k)\}$ holds. Symmetrically, a min-node i with two children j and k is called (σ,τ)-switchable if $val_{\sigma,\tau}(i) > \mathsf{min}\{val_{\sigma,\tau}(j), val_{\sigma,\tau}(k)\}$ holds. Intuitively, this means that at a node a local improvement can be achieved if the other edge instead of the choosen one (by the strategy σ or τ) is used. A max-strategy σ' id obtained from a max-strategy σ by *switching* a switchable max-node i with two children by $(i,j) \in \sigma' \Leftrightarrow_{df} (i,j) \notin \sigma$. The intuitive meaning is to look for a local improvement at node i by choosing a 'better' edge than the initially chosen one. The switching of a min-node is defined analogously.

10.3.1 Successive Approximation

The first algorithm we will consider is the successive approximation algorithm, due to [Sha53], here given as Algorithm 12. It is no exact algorithm but an approximation algorithm which computes a sequence of feasible node labellings which converges towards the optimal value function. Its main drawback, as pointed out in [Con], is that its convergence rate can be exponentially slow in $|V|$.

The most operations are rather easy to implement; we will shortly dwell on the only complex statement in Line 8. However, by definition of feasability this leads to a simple system of linear equations which can be solved in $\Theta(|V|^3)$ time by Gaussian elimination. The **until**-loop between Line 9 and Line 28 takes $\Theta(|V|)$ time in every reasonable implementation, and the same holds for the part between Line 1 and 7. Hence the overall runtime for k iterations including Lines 1 till 8 is in $\Theta(|V|^3 + k|V|)$. So the efficiency of Algorithm 11 using Algorithm 12 depends on the number of iterations of Algorithm 12 in Line 2.

10.3.2 Policy Improvement

As indicated by its name, this algorithm iteratively tries to improve policies till it stops with a pair of optimal policies as shown in Algorithm 13 which was proposed in [HK66]. In contrast to the approximation Algorithm 12, this is an exact algorithm.

For this algorithm, no lower bound for the numbers of executions of the **until**-loop is known. In [Con] an upper bound of $2^{|V_{\mathsf{max}}|-f(|V_{\mathsf{max}}|)} + 2^{o(f(|V_{\mathsf{max}}|))}$ for any function $f(n) \in o(n)$ is shown.

To obtain rigorous lower bounds we concentrate on a variant of SSGs which contain no max-nodes. In [How66] a proof for the correctness of Algorithm 14 under these circumstances is given. Note that due to the absence of max-nodes there is exactly

Algorithm 12 Successive Approximation

1: $\overline{val}(sink_0) \leftarrow 0$, $\overline{val}(sink_1) \leftarrow 1$
2: **for all** $v \in V_{\mathsf{min}}$ **do**
3: $\quad\overline{val}(v) \leftarrow 0$
4: **end for**
5: **for all** $v \in V_{\mathsf{max}}$ **do**
6: $\quad\overline{val}(v) \leftarrow 1$
7: **end for**
8: determine $\overline{val}(v)$ for all average nodes v such that $\overline{val}$ is feasible while $\overline{val}$ remains unchanged on $\{sink_0, sink_1\} \cup V_{\mathsf{min}} \cup V_{\mathsf{max}}$
9: **repeat**
10: $\quad\overline{val'}(sink_0) = 0$
11: $\quad\overline{val'}(sink_1) = 1$
12: $\quad$**for all** $i \in V - \{sink_0, sink_1\}$ **do**
13: $\quad\quad$**if** i has one child j **then**
14: $\quad\quad\quad\overline{val'}(i) \leftarrow \overline{val}(j)$
15: $\quad\quad$**else**
16: $\quad\quad\quad$**if** i is a min node with two children j and k **then**
17: $\quad\quad\quad\quad\overline{val'}(i) = \min\{overlineval(j), \overline{val}(k)\}$
18: $\quad\quad\quad$**end if**
19: $\quad\quad\quad$**if** i is a max node with two children j and k **then**
20: $\quad\quad\quad\quad\overline{val'}(i) = \max\{overlineval(j), \overline{val}(k)\}$
21: $\quad\quad\quad$**end if**
22: $\quad\quad\quad$**if** i is an $\mathsf{average}$ node with two children j and k **then**
23: $\quad\quad\quad\quad\overline{val'}(i) = \frac{1}{2}(overlineval(j) + \overline{val}(k))$
24: $\quad\quad\quad$**end if**
25: $\quad\quad$**end if**
26: $\quad$**end for**
27: $\quad\overline{val} \leftarrow \overline{val'}$
28: **until** $\overline{val}$ is stable

one max-strategy, namely the empty one. This explains the criterion of switchability in Line 3 and the stopping condition of the **until**-loop in Line 5. For brevity, an SSG without max-nodes is called a *min-SSG*.

Algorithm 13 Hoffman-Karp Algorithm

1: choose an arbitrary pair (σ, τ) of max- and min
2: **repeat**
3: switch all (σ, τ)-switchable max-nodes to obtain the max-strategy σ'
4: compute a min-strategy τ' which is optimal with respect to σ'
5: $(\sigma, \tau) \leftarrow (\sigma', \tau')$
6: **until** $val_{\sigma,\tau}$ is stable

Algorithm 14 Policy Improvement Algorithm for min-SSGs

1: choose an arbitrary min-strategy τ
2: **repeat**
3: switch an arbitrary $(\emptyset, \tau)$-switchable min-node to obtain the min-strategy τ'
4: $\tau \leftarrow \tau'$
5: **until** $val_{\emptyset,\tau}$ is stable

In [MC94] an exponential lower bound for the number of iterations of the Policy Improvement Algorithm is shown. We will use the results from there to construct a family of min-SSGs for which Algorithm 11 offers a running time advantage compared to the immediate application of Algorithm 14.

The running time of the Policy Improvement Algorithm depends on how the node in Line 3 is chosen. However, for almost all not-randomised implementations there are examples with an exponential number of iterations. We will do a generic construction for all kinds of implementations considered in [MC94]. For every implementation, there is a family $(M_n)_{n\in\mathbb{N}}$ of min-SSGs such that Algorithm 14 performs at least $c^{|V_n|}$ iterations for some constant $c > 1$ (implicitly, we used the notation $M_n =_{df} (((V_n, E_n), g_n), a_n)$). Moreover, these families can be constructed in such a way that $|V_n| = n$ and $|E_n| \in \mathcal{O}(n)$ hold.

We now construct a family $(M'_n)_{n\in\mathbb{N}}$ of min-SSGs as follows (canonically, we have $M'_n =_{df} (((V'_n, E'_n), g'_n), a'_n)$):

1. $(V'_n)_{\mathsf{min}} =_{df} \bigcup_{i=1}^{n} (V_n)^i_{\mathsf{min}}$
2. $(V'_n)_{\mathsf{average}} =_{df} \bigcup_{i=1}^{n} (V_n)^i_{\mathsf{average}}$

3. $(v, w) \in E_n \Rightarrow (v^i, w^i) \in E'_n$ for all $w \notin \{sink_{0n}, sink_{1n}\}$ and $i \in \{1 \dots n\}$
4. $(v, sink_{0n}) \in E_n \Rightarrow (v^i, sink'_{0n}) \in E'_n$ for all $i \in \{1 \dots n\}$
5. $(v, sink_{1n}) \in E_n \Rightarrow (v^i, sink'_{1n}) \in E'_n$ for all $i \in \{1 \dots n\}$

This means that M'_n consists of n copies of M_n which share two common sink nodes $sink'_{0n}$ and $sink'_{1n}$. Clearly, the equalities $|V'_n| = n \cdot |V_n| - 2n + 2 = n^2 - 2n + 2$ hold by construction.

The analysis' in [MC94] can easily transferred to the family M'_n. The crucial point is that the associated graphs of each M'_n consist of n strongly connected components and the two sink nodes which have only ingoing edges. So there is no interference between the several strongly connected components (an argument which is also used in [MC94]) and every such strongly connected component requires an exponential number (in the number of its nodes) of iterations. Hence the execution of the Policy Improvement Algorithm on M'_n takes $n \cdot c^n$ iterations. However, there is a bisimulation equivalence B such that M'_n/B is isomorphic to M_n (this is obvious because of the construction of M'_n) so the execution of the Policy Improvement Algorithm on M'_n takes at most c^n iterations. The computation of the coarsest bisimulation for M_n takes $\mathcal{O}(n \cdot log(n))$ time, so in this case Algorithm 11 is preferable to the immediate application of the Policy Improvement Algorithm.

Chapter 11

An Algebraic Approach

In this chapter we give an approach to bisimulations via the theory of semirings and dioids. We give first an introduction into semirings and study how partitions, equivalence relations and equivalence classes can be handled in the semiring world. Finally, we define bisimulations in this setting and show an application to a liveness control objective. Most of the material is based on [GMS09].

11.1 Overview

Semirings have been long in use for describing graphs and relations (see e.g. [Car80]). Also edge-weighted graphs were treated successfully in this framework using fuzzy relations, as in [GM08a] and [Kaw06]. In [Win08] bisimulations are described in category theory, whereas [Pou07] gives a lattice-theoretic abstraction of bisimulations. However, no description of bisimulations and equivalences using semirings and dioids has been done before.

The results (which not only cover the theory of bisimulations) can be applied to other structures like path and tree algebras which can be described by semirings. Another advantage of this approach is that the amount of higher order logic is reduced and only a one sorted algebra is used. Therefore this formulation is better suitable for the application of automated proof systems. This will be done in the following Chapter 12.

11.2 Semirings, Tests and Partitions

Two basic operations in relation algebra are union and composition of relations. These two operations and their interplay is captured in an abstract way in the definition of a semiring.

Definition 11.2.1

1. An *idempotent semiring* is a structure $S = (M, +, 0, \cdot, 1)$ such that $0 \neq 1$, $(M, +, 0)$ and $(M, \cdot, 1)$ are monoids, choice $+$ is commutative and idempotent, and composition $\cdot$ distributes through $+$ and 0 is an annihilator with respect to composition, i.e. $a \cdot 0 = 0 = 0 \cdot a$ for all $a \in M$.

2. The *natural order* $\sqsubseteq$ is given by $x \sqsubseteq y \Leftrightarrow_{df} x + y = y$.

3. An element $x \in N$ of a subset $N \subseteq M$ is called *atomic in* N if $x \neq 0$ and $\forall\, y \in N : y \neq 0 \wedge y \sqsubseteq x \Rightarrow y = x$.

4. A subset $N \subseteq M$ is called *atomic* if every element $x \in N$ is the supremum of the atoms of N below x.

The operator $+$ is also called the *sum* operator, and $\cdot$ the *multiplication* operator.

So an idempotent semiring is almost the same as a dioid (cf. Chapter 8); a semiring lacks only the existence of arbitrary suprema.

The element 0 is $+$-irreducible in an idempotent semiring, i.e. $x + y = 0 \Leftrightarrow x = 0 = y$ holds for all $x, y \in M$. More general, we also have $\sum_{x \in N} x = 0 \Leftrightarrow \forall x \in N : x = 0$.

Another type of dioid we did not consider in Chapter 8 is given for example by $(\mathsf{Rel}(M), \cup, \emptyset, ;, \mathsf{id}_M)$ where M is an arbitrary set and $\mathsf{Rel}(M)$ denotes the set of all relations over M. Union of relation corresponds to semiring addition, and composition to multiplication. An element is atomic in $\mathsf{Rel}(M)$ iff it is a relation consisting of exactly one element. So every relation over M is atomic in $(\mathsf{Rel}(M), \cup, \emptyset, ;, \mathsf{id}_M)$ because is can be written as the union of singleton relations. Of course, elements from this dioid will rarely serve as edge labels.

As a running example for this Chapter we use a simple non-deterministic unlabelled transition system whose state is described by a variable which can have values from the interval $[0, 9]$ (so it can be viewed as an element of the semiring $(\mathsf{Rel}([0, 9]), \cup, \emptyset, ;, \mathsf{id}_{[0,9]})$, to which we will also refer in some examples). Its transition relation is given by the following relation T which describes the relation between an input state x and an output state y:

$xTy \Leftrightarrow_{df} (x \in [0,2] \wedge y = x+4) \vee (x \in [4,6] \wedge y = x+3) \vee (x \in [4,6] \wedge y = x-4)$.

So semiring in the sense of Definition 11.2.1 can serve as an abstract description of union and composition of relations over a fixed set. However, in this version we are not yet in the situation to reason about subsets of the carrier set of relations or states of a transition system. To resolve this problem in [MB85] the concept of tests was introduced.

Definition 11.2.2 Let $S = (M, +, 0, \cdot, 1)$ be an idempotent semiring. The set of *tests* of S, denoted by $\mathsf{test}(S)$, is the maximal Boolean subalgebra of S of elements below 1 with $+$ as join and $\cdot$ as meet. We denote the complement of a test p by $\neg p$; so $\neg p$ is the unique element fulfilling $p + \neg p = 1$ and $p \cdot \neg p = 0$. The set of atomic test of S, i.e. the set of atomic elements of $\mathsf{test}(S)$, is denoted by $\mathsf{atest}(S)$.

Usually, test are denoted by $p, q, r, \ldots$ and primed and indexed variations of them. In every idempotent semiring the elements 0 and 1 are tests. On $\mathsf{test}(S)$ multiplication and infimum coincide, so we have $p \leq q \Leftrightarrow p \cdot q = p$ for all tests p and q. In particular, multiplication is commutative on $\mathsf{test}(S)$.

In our example $(\mathsf{Rel}([0,9]), \cup, \emptyset, ;, \mathsf{id}_{[0,9]})$ the tests are exactly all subrelations of $\mathsf{id}_{[0,9]}$, in particular $\emptyset$ and $\mathsf{id}_{[0,9]}$ are tests (they correspond to 0 and 1, respectively). Moreover, each subrelation of $id[0,9]$ can be identified in a canonical way with a subset of $[0,9]$ and vice versa. So they can be used to model subsets without introducing a new sort. The set of tests is in this case atomic, and the atoms of $\mathsf{test}(S)$ are here given by $\mathsf{atest}(S) = \{\{(x,x)\} \mid x \in [0,9]\}$.

From now on we assume in this chapter that the set $\mathsf{test}(S)$ is atomic for every semiring S under consideration. The atomic tests in semirings representing relations or transition systems atomic tests correspond to single elements or nodes of a graph.

In this chapter we will often consider the supremum of all elements contained in a subset of $\mathsf{test}(S)$. To avoid notational overflow we introduce the abbreviation $\sum P =_{df} \sum_{p \in P} p$ for finite $P \subseteq \mathsf{test}(S)$ (cf. the notation of the supremum in Definition 8.1.1). So, if the semiring under consideration is also a dioid the meaning of $\sum$ remains the same.

We note a simple fact which follows from the $+$-irreducibility of 0 by contraposition:

Lemma 11.2.1 *Let $S = (M, +, 0, \cdot, 1)$ be an idempotent semiring and $P \subseteq M$ an arbitrary subset. Then $\sum P \neq 0 \Leftrightarrow \exists\, p \in P : p \neq 0$ holds.*

11.3 Partitions

In this section we will define the concept of a partition in the setting of semirings and investigate it in this framework. Remember the definition of a partition (cf. Chapter 2) and the fact that union corresponds to the supremum and multiplication on $\mathsf{test}(S)$ to the infimum, i.e. intersection. Moreover, 0 models the empty set and 1 the carrier set, so the following definition is motivated:

Definition 11.3.1 A finite subset $P \subseteq \mathsf{test}(S)$ is called a *partition* if $\sum P = 1$ and for all $p, q \in P$ the equivalence $p \cdot q = 0 \Leftrightarrow p \neq q$ holds.

Now the term partition is overloaded, once in the sense of Definition 11.3.1 and once in the sense of partition of a set. To distinguish between them we will use the term *partition* in the sense of Definition 11.3.1 and use *set partition* if we mean the partition of a set.

So in every semiring the set $\{1\}$ forms a partition, and, because we assume that in every semiring S under consideration the set of tests is atomic, the set of atomic tests $\mathsf{atest}(S)$ is also a partition. Clearly, a partition contains only tests different from 0 unless $0 = 1$ holds.

If a set can be written as the union of sets from a fixed set partition then the choice of these sets is unique. An analogous property about partitions is given by next lemma:

Lemma 11.3.1 *Let P be a partition and $P', P'' \subseteq P$ subsets of it. Then the equivalence $\sum P' = \sum P'' \Leftrightarrow P' = P''$ holds.*

Proof: The direction '$\Leftarrow$' is obvious. So assume for the sake of contradiction that there are two different subsets $P', P'' \subseteq P$ with $\sum P' = \sum P''$. W.l.o.g there is a $p \in P'$ with $p \notin P''$ and $p \neq 0$. First we calculate:

$$
\begin{aligned}
& p \cdot \textstyle\sum P' = \\
& \quad \{ \text{ choice of } p \ \} \\
& p \cdot \textstyle\sum(\{p\} \cup (P' - \{p\})) = \\
& \quad \{ \text{ splitting of } \textstyle\sum \ \} \\
& p \cdot (\textstyle\sum\{p\} + \sum(P' - \{p\})) = \\
& \quad \{ \text{ distributivity } \} \\
& p \cdot \textstyle\sum\{p\} + p \cdot \sum(P' - \{p\}) = \\
& \quad \{ \text{ supremum of singleton set } \} \\
& p \cdot p + p \cdot \textstyle\sum(P' - \{p\}) = \\
& \quad \{ \text{ distributivity, idempotency of } \cdot \text{ on } \mathsf{test}(S) \ \} \\
& p + \sum_{p' \in (P' - \{p\})} (p \cdot p') =
\end{aligned}
$$

$$
\begin{array}{l}
\quad \{ \text{ partition properties } \} \\
p + \sum\limits_{p' \in (P' - \{p\})} 0 = \\
\quad \{ \text{ obvious } \} \\
p
\end{array}
$$

On the other hand we have the following:

$$
\begin{array}{l}
p \cdot \sum P'' = \\
\quad \{ \text{ distributivity } \} \\
\sum\limits_{p'' \in P''} (p \cdot p'') = \\
\quad \{ \forall p'' \in P'' : p \neq p'' \text{, partition properties } \} \\
\sum\limits_{p'' \in P''} 0 = \\
\quad \{ \text{ obvious } \} \\
0
\end{array}
$$

This means that $p \cdot \sum P' \neq p \cdot \sum P''$, which contradicts the assumption $\sum P' = \sum P''$. ■

The following lemma gives an easy but often used connection between subsets of a partition and their complements:

Lemma 11.3.2 *Let $P' \subseteq P$ be a subset of a partition P. Then the equality $\neg \sum P' = \sum(P - P')$ holds. In particular, for $p \in P$ we have $\neg p = \sum(P - \{p\})$.*

Proof: We have $\sum(P - P') + \sum P' = \sum((P - P') \cup P') = \sum P = 1$ due to the properties of the supremum and the the fact that P is a partition. On the other hand we have $\sum(P - P') \cdot \sum P = \sum\limits_{p \in (P - P'), p' \in P'} (p \cdot p')$. Because all p and p' in the products of the sum are different all these products become 0, and so does the whole sum. This shows that $\sum(P - P')$ fulfills the properties for being the complement of $\neg \sum P'$. The second claim is simply a special case of the first one for singleton sets. ■

Partitions in our running example are (among many others) e.g. $\{\mathsf{id}_{[0,6]}, \mathsf{id}_{]6,9]}\}$ and $\{\mathsf{id}_{[0,9] \cap \mathbb{Q}}, \mathsf{id}_{[0,9] - \mathbb{Q}}\}$.

The refinement of partitions plays an important role. For instance, algorithms for computing the coarsest bisimulation work by successive refinement of an initial partition. This term is captured in the next definition:

Definition 11.3.2 We say that partition Q *refines* partition P, denoted by $Q \leq_r P$, if every element of P can be written as the sum of suitable elements of Q. When $Q \leq_r P$ we say also that P is *coarser* than Q. Clearly, $\leq_r$ is an order on the set of partitions of a fixed semiring.

Note that the subset $Q' \subseteq Q$ with $\sum Q' = p$ for any p is unique due to Lemma 11.3.1.

In the context of set partitions one might expect a slightly different definition of partition refinement. Given two set partitions $\mathcal{V} = \{V_i \mid i \in I\}$ and $\mathcal{V}' = \{V'_j \mid j \in J\}$ of a set V a natural definition would be that $\mathcal{V}$ refines $\mathcal{V}'$ if for every $V_i \in \mathcal{V}$ there is a $V'_j \in \mathcal{V}'$ with the property $V_i \subseteq V'_j$. However, if we translate this definition into the language of semirings it turns out that this equivalent to Definition 11.3.2:

Lemma 11.3.3 *Let P and Q be two partitions. Then Q refines P iff for every $q \in Q$ there is a $p \in P$ with $q \leq p$.*

Proof: We show this equivalence using some subsequent Lemmas (note that these Lemmas use only Definition 11.3.2. We decided to prove Lemma 11.3.3 before to clarify the nearby relation to the other possible definition).

$\Rightarrow$: Let P and Q be partitions with $Q \leq_r P$, and pick an arbitrary $q \in Q$. Due to $0 \neq q = q \cdot 1 = q \cdot \sum P = \sum_{p \in P} q \cdot p$ there has to be a $p \in P$ with $q \cdot p \neq 0$. But for such a p we have $q \leq p$ because of Lemma 11.3.4.

$\Leftarrow$: Consider two partitions P and Q such that for every $q \in Q$ there is a $p \in P$ with $q \leq p$, and fix arbitrary $q \in Q$ and $p \in P$ with $q \leq p$. First, this implies $q \cdot p = q$ (note that on tests multiplication and infimum coincide), and hence $q \cdot p \neq 0$, since q has to be different from 0. Second, assume that there is a $p' \in P$ with $p' \neq p$ and $q \cdot p' \neq 0$. Then we have both $q = q \cdot p$ and $q = q \cdot p'$. Combining these two equations yields $q = q \cdot p \cdot p'$, and because of $p \cdot p' = 0$ we have $q = 0$, which contradicts the definition of a partition. So for every $q \in Q$ there is a unique $p \in P$ with $q \cdot p \neq 0$, and the claim holds according to Lemma 11.3.5. ■

When dealing with partitions of sets and their refinement an obvious observation is that if a set of the finer partition has a nonempty intersection with a set of the coarser partition it has to be fully contained in this set. This property is stated in the next lemma in our semiring setting:

Lemma 11.3.4 *Let P and Q be partitions with $Q \leq_r P$. Then for all $q \in Q$ and $p \in P$ the implication $q \cdot p \neq 0 \Rightarrow q \leq p$ holds.*

Proof: Because of $Q \leq_r P$ there is a subset $Q' \subseteq Q$ with $p = \sum Q'$. Due to distributivity we have $q \cdot p = q \cdot \sum Q' = \sum_{q' \in Q'} q \cdot q'$. Now 0 is $+$-irreducible, so there has to be a $q' \in Q'$ with $q \cdot q' \neq 0$. Since Q is a partition therefrom $q = q'$ follows and hence $q \cdot q' = q$ and eventually $q \leq p$. ■

The next lemma reflects the fact that every set of a partition of a set is contained in exactly one set of a coarser partition.

Lemma 11.3.5 *A partition Q refines a partition P iff for all $q \in Q$ there is a unique $p \in P$ with $q \cdot p \neq 0$.*

Proof: '$\Rightarrow$': First we show that for every $q \in Q$ there is a $p \in P$ with $q \cdot p \neq 0$. So consider the equation chain $q = q \cdot 1 = q \cdot (\sum P) = \sum_{p \in P}(q \cdot p)$. Because of $q \neq 0$ and $+$-irreducibility of 0 there has to be a $p \in P$ with $q \cdot p \neq 0$.
Assume now that there are two different $p, p' \in P$ with $q \cdot p \neq 0$ and $q \cdot p' \neq 0$. Then Lemma 11.3.4 yields both $q \leq p$ and $q \leq p'$. So we have $0 \neq q = q \cdot q \leq p \cdot p'$, which contradicts $p \neq p'$ (in this case $p \cdot p' = 0$ had to hold). Hence p is unique.
'$\Leftarrow$': Consider an arbitrary $p \in P$ and chose a subset $Q' \subseteq Q$, defined by $Q' = \{q \in Q \mid q \cdot p \neq 0\}$. We will show that $p = \sum Q'$ holds. First we consider the equation chain $p = (\sum Q) \cdot p = (\sum Q' + \sum(Q - Q')) \cdot p = (\sum Q') \cdot p$. Because on tests multiplication and infimum coincide this implies $p \leq \sum Q'$.
The reverse inequality $\sum Q' \leq p$ is equivalent to $\forall\, q \in Q' : q \leq p$. Assume now that there is a $q \in Q'$ with the property $q \not\leq p$. By Lemma 11.3.2 this implies $0 \neq q \cdot \neg p = q \cdot \sum(P - \{p\}) = \sum_{p' \in P - \{p\}} q \cdot p'$. Because 0 is $+$-irreducible there has to be a $p' \in P - \{p\}$ such that $q \cdot p' \neq 0$. But this is a contradiction to the assumption that p is unique with the property $q \cdot p \neq 0$. ■

Lemma 11.3.6 *Let P and Q be partitions with $Q \leq_r P$ and assume $p \in P$ and $p = \sum Q'$ for some $Q' \subseteq Q$. Then for all $q \in Q$ we have $p \cdot q \neq 0 \Leftrightarrow q \in Q'$.*

Proof: '$\Rightarrow$': Because of $q \cdot p = q \cdot \sum_{q' \in Q'} q' = \sum_{q' \in Q'}(q \cdot q') \neq 0$ there has to be a $q' \in Q'$ with $q \cdot q' \neq 0$. From the definition of a partition it follows that $q = q'$ and hence $q \in Q'$ hold.
'$\Leftarrow$': Fix an arbitrary $q \in Q'$. Then we have $q \leq p$ and therefore $p \cdot q = q$. Because Q is a partition we have $q \neq 0$, and the claim is shown. ■

A basic fact is that an equivalence relation relates only elements from the same equivalence class. On the other hand, the equivalence classes form a partition of the carrier set of an equivalence relation. This behaviour is captured in the semiring setting by the following definition:

Definition 11.3.3 An element $r \in M$ *respects* a partition P if $r = \sum_{p \in P} p \cdot r \cdot p$.

Intuitively, a relation which respects a set partition can not relate elements from different sets of the partition. This follows already from the previous definition:

Lemma 11.3.7 *Let $r \in M$ respect the partition P and let $p, p' \in P$ such that $p \neq p'$. Then $p \cdot r \cdot p' = 0$.*

Proof: For arbitrary $p'' \in P$ we know from the definition of a partition that $p \cdot p'' = 0$ or $p' \cdot p'' = 0$ holds. So we can calculate:

$$
\begin{aligned}
& p \cdot r \cdot p' = \\
& \quad \{\ r \text{ respects } P\ \} \\
& p \cdot (\sum_{p'' \in P} p'' \cdot r \cdot p'') \cdot p' = \\
& \quad \{\ \text{distributivity}\ \} \\
& \sum_{p'' \in P} p \cdot p'' \cdot r \cdot p'' \cdot p' = \\
& \quad \{\ \text{above remark}\ \} \\
& \sum_{p'' \in P} 0 = \\
& \quad \{\ \text{trivial}\ \} \\
& 0,
\end{aligned}
$$

and we are done. ■

The next corollary is an easy consequence:

Corollary 11.3.1 *Let $r \in M$ respect the partition P. Then for all $p \in P$ the equalities $p \cdot r \cdot \neg p = 0 = \neg p \cdot r \cdot p$ hold.*

Proof: We show that in every semiring as in Definition 11.2.1 for every test the inequality $p \neq \neg p$ holds. Assume a test p with $p = \neg p$. On the one hand we have $p + \neg p = 1$. By assumption this leads to $p + p = 1$ which implies $p = 1$. In the other hand, from $p \cdot \neg p = 0$ follows $p \cdot p = 0$ and therefrom $p = 0$. But this is a contradiction to $0 \neq 1$ in Definition 11.2.1.
Now we can apply Lemma 11.3.7, because $p \neq \neg p$ holds and $\{p, \neg p\}$ forms a partition. ■

If a relation respects a set partition it also respects every coarser set partition. This important fact is stated in the following theorem:

Theorem 11.3.1 *Let partition Q refine partition P. If $r \in M$ respects Q then it respects P, too.*

Proof: For arbitrary $p \in P$ we introduce the abbreviation $Q_p \subseteq Q$ for the unique (cf. Lemma 11.3.1) subset of Q with the property $\sum Q_p = p$.
First we show that $\bigcup_{p \in P} Q_p = Q$ holds. Trivially we have $\bigcup_{p \in P} Q_p \subseteq Q$, so assume there is a q with $q \in Q$ and $q \notin \bigcup_{p \in P} Q_p$. By construction, $\bigcup_{p \in P} Q_p$ is a partition, and we can reason as follows:

$$
\begin{aligned}
& \neg q = \\
\end{aligned}
$$

$$
\begin{array}{l}
\quad \{ \text{ Lemma 11.3.2 } \} \\
\sum_{q' \in Q, q' \neq q} q' \geq \\
\quad \{ \ q \notin \bigcup_{p \in P} Q_p, \ \bigcup_{p \in P} Q_p \subseteq Q \ \} \\
\sum_{q' \in Q, q' \neq q} q' = \\
\quad \{ \ \bigcup_{p \in P} Q_p \text{ is partition } \} \\
1
\end{array}
$$

This means $\neg q \geq 1$, which implies $q \leq 0$ and finally $q = 0$, which contradicts the fact that Q is a partition.
After this preliminary deliberation we can now calculate:

$$
\begin{array}{l}
\sum_{p \in P} p \cdot r \cdot p = \\
\quad \{ \text{ Construction of } Q_p \ \} \\
\sum_{p \in P} ((\sum Q_p) \cdot r \cdot (\sum Q_p) = \\
\quad \{ \text{ Distributivity } \} \\
\sum_{p \in P} (\sum_{q,q' \in Q_p} (q \cdot r \cdot q')) = \\
\quad \{ \text{ splitting of } \sum \ \} \\
\sum_{p \in P} ((\sum_{q \in Q_p} q \cdot r \cdot q) + (\sum_{q,q' \in Q_p, q \neq q'} q \cdot r \cdot q')) = \\
\quad \{ \text{ Lemma 11.3.7 } \} \\
\sum_{p \in P} (\sum_{q \in Q_p} q \cdot r \cdot q) = \\
\quad \{ \ \bigcup_{p \in P} Q_p = Q \ \} \\
\sum_{q \in Q} q \cdot r \cdot q = \\
\quad \{ \ r \text{ respects } Q \ \} \\
r,
\end{array}
$$

and the proof is finished. ■

11.4 Modality and Symmetry

In the previous section we showed how partitions can be handled in the abstract setting of semirings. However, we did not reason about equivalence relations. The reason is that the operations we have at our disposal till now can model only composition, union and restriction of relations. To reason about equivalence relations we need a concept how we can model also the converse of a relation and characterise symmetry of relations. As a first step we introduce modal operators which enable us to reason about image and preimage of relations.

Definition 11.4.1 A *modal (idempotent) semiring* (see e.g. [DMS06]) is a structure $(M, +, 0, \cdot, 1, |\cdot\rangle, \langle\cdot|)$ where $(M, +, 0, \cdot, 1)$ is an idempotent semiring and the forward and backward diamond operators $|\cdot\rangle, \langle\cdot| : M \to (\mathsf{test}(S) \to \mathsf{test}(S))$ are axiomatised as follows for all $x, y \in M$ and $p, q \in \mathsf{test}(S)$:

1. $|x\rangle q \leq \neg p \Leftrightarrow p \cdot x \cdot q \leq 0 \Leftrightarrow \langle x|p \leq \neg q$
2. $|x\rangle(|y\rangle q) = |x \cdot y\rangle q$ and $\langle x|(\langle y|q) = \langle y \cdot x|q$

In the concrete semiring of relations over a fixed set, the backward diamond $\langle x|q$ and the forward diamond $|x\rangle q$ correspond to the image and preimage, resp. of the subset characterised by q under the relation x. So in our example we have $|T\rangle\mathsf{id}_{[7,8]} = \mathsf{id}_{[4,5]}$ and $\langle T|\mathsf{id}_{[5,6[} = \mathsf{id}_{[1,2[} \cup \mathsf{id}_{[8,9[}$.
The following lemma lists some properties of the diamond operators without proof (the proofs can be found e.g. in [DMS06] or [DMS11]):

Lemma 11.4.1 *Let $x, y \in M$ and $p, q \in \mathsf{test}(S)$ be arbitrary. Then the following hold:*

1. $|p\rangle q = p \cdot q = \langle p|q$.
2. $|x\rangle 0 = 0 = \langle x|0$.
3. $|0\rangle p = 0 = \langle 0|p$.
4. $|x + y\rangle p = |x\rangle p + |y\rangle p$.
5. $|x\rangle(p + q) = |x\rangle p + |x\rangle q$.
6. $x \leq y \wedge p \leq q \Rightarrow |x\rangle p \leq |y\rangle q$.

All the properties of the forward diamond given here hold also for the backward diamond in a symmetric way.
The box operators are defined via de Morgan duality by $|x]p = \neg|x\rangle\neg p$ and $\langle x|p = \neg\langle x|\neg p$. In the relational setting, the forward box $|R]N$ corresponds to the set of elements from which every R-step leads inevitably into N. Note that nodes without successors belong automatically to $|R]N$.
Diamond and Box form a Galois connection and enjoy the following properties (see [DMS11]):

Lemma 11.4.2 *Let a be arbitrary and consider an arbitrary test p. Then the following inequalities and equivalences hold:*

1. $|a\rangle\langle a|p \leq p \leq \langle a||a\rangle p$,
2. $\langle a||a]p \leq p \leq |a]\langle a|p$,
3. $\langle a|p \leq p \Leftrightarrow p \leq |a]p$ *and* $|a\rangle p \leq p \Leftrightarrow p \leq \langle a|p$.

In a dioid the equations 4 and 5 from Lemma 11.4.1 can be strengthened from suprema of binary sets to suprema of arbitrary sets, i.e. in a dioid $|\sum_{x\in N} x\rangle p = \sum_{x\in N} |x\rangle p$ and $|x\rangle \sum_{p\in P} p = \sum_{p\in P} |x\rangle p$ hold for all subsets $N \subseteq M$ and $P \subseteq \mathsf{test}(S)$.

As a consequence of Part 1 of Definition 11.4.1 we have $p \cdot |x\rangle q \leq 0 \Leftrightarrow p \cdot x \cdot q \leq 0 \Leftrightarrow q \cdot \langle x|p \leq 0$. This implies the following lemma:

Lemma 11.4.3 *Let p be an atomic test. Then for all $x \in M$ and all tests q the equivalence $p \cdot x \cdot q \neq 0 \Leftrightarrow p \leq |x\rangle q$ holds.*

Proof: By contraposition of the above equivalence, we get $p \cdot x \cdot q \neq 0 \Leftrightarrow p \cdot |x\rangle q \neq 0$. Because of $|x\rangle q \leq 1$ we have $p \cdot |x\rangle q \leq p$, and since p is atomic this is equivalent to $p \cdot |x\rangle q = p$. But this is the case iff $p \leq |x\rangle q$ holds. ■

The symmetric property holds for the backward diamond, i.e. for an atomic test p and arbitrary $x \in M$ and $q \in \mathsf{test}(S)$ we have the equivalence $q \cdot x \cdot p \neq 0 \Leftrightarrow p \cdot \langle x|q \neq 0$.

Often, we do not want to compare single tests but also functions on test. Therefore we overload the $\leq$-sign:

Definition 11.4.2 Let $f : \mathsf{test}(S) \to \mathsf{test}(S)$ and $g : \mathsf{test}(S) \to \mathsf{test}(S)$ be functions on $\mathsf{test}(S)$. Then we define the order pointwise by

$$f \leq g \Leftrightarrow_{df} \forall p \in \mathsf{test}(S) : f(p) \leq g(p)$$

In an analogous way $f = g$ is defined.

So $\langle x| \leq \langle y|$ stands for $\forall p \in \mathsf{test}(S) : \langle x|p \leq \langle y|p$ and $\langle x| = \langle y|$ for $\forall p \in \mathsf{test}(S) : \langle x|p = \langle y|p$. Clearly, the above defined relation is an order and hence reflexive, transitive and antisymmetric.

The symmetry of a relation R is expressed via $R^\circ \subseteq R$ or the equivalent formulation $R^\circ = R$. A pure semiring as given in Definition 11.2.1 offers no possibility to model symmetry since it lacks the concept of a converse operation. So we have to look for a different way to handle symmetry.

Let us for an intermediate step assume that in a semiring there is for every $x \in M$ a virtual converse x°. Then this operation should certainly fulfill the equivalence

$p \cdot x^\circ \cdot q = 0 \Leftrightarrow q \cdot x \cdot p = 0$ for all $p, q \in \mathsf{test}(S)$. By Part 1 of Definition 11.4.1 this is equivalent to $|x^\circ\rangle = \langle x|$ and $\langle x^\circ| = |x\rangle$. This means we can avoid the explicit definition of a converse operation by considering the behaviour of the forward and backward diamond of an element on all tests. This idea gives rise to the next definition:

Definition 11.4.3 An element x of a modal semiring is called *symmetric* if $\langle x| = |x\rangle$.

From Part 4 of Lemma 11.4.1 it follows immediately that in a dioid the supremum of arbitrary symmetric elements is also symmetric. In particular, this holds also for finite sums in semirings.

Our example relation is not symmetric in the sense of Definition 11.4.3, because we have $|T\rangle\{(5,5)\} = \{(1,1)\}$, but $\langle T|\{(5,5)\} = \{(1,1),(8,8)\}$. However, the restricted relation $T' = T\,;(\mathsf{id}_{[0,2]} \cup \mathsf{id}_{[4,6]})$ is symmetric in this sense, as can be easily verified.

Next we consider a certain class of semirings which contain a greatest element $\top$ with $x \leq \top$ for all $x \in M$. In the semiring of relations over a fixed set the greatest element is the universal relation over this set. Also, this semiring fulfills the following definition:

Definition 11.4.4 Let $S = (M, +, 0, \cdot, 1, |\cdot\rangle, \langle\cdot|)$ be a modal semiring with greatest element $\top$. We say that S satisfies the *Tarski rule* if $x \neq 0 \Leftrightarrow \top \cdot x \cdot \top = \top$ holds for all $x \in M$.

The Tarski rule has also an equivalent formulation:

Lemma 11.4.4 *Let $S = (M, +, 0, \cdot, 1, |\cdot\rangle, \langle\cdot|)$ be a modal semiring with greatest element $\top$. S satisfies the Tarski rule iff $\top \cdot x \cdot \top = \top \cdot y \cdot \top \Leftrightarrow (x = 0 \Leftrightarrow y = 0) \Leftrightarrow (x \leq 0 \Leftrightarrow y \leq 0)$ holds for all $x, y \in M$.*

Proof: Clearly, the Tarski rule follows from the condition of Lemma 11.4.4 if we choose x arbitrarily and set $y = 1$.
So assume now first $x = 0 \wedge y = 0$. This implies $\top \cdot x \cdot \top = 0 = \top \cdot y \cdot \top$. Also, $x \neq 0 \wedge y \neq 0$ implies $\top \cdot x \cdot \top = \top = \top \cdot y \cdot \top$ in presence of the Tarski rule. This shows $\top \cdot x \cdot \top = \top \cdot y \cdot \top \Leftarrow (x = 0 \Leftrightarrow y = 0)$. For the other direction assume $\top \cdot x \cdot \top = \top \cdot y \cdot \top$ and $x = 0$. This implies $\top \cdot x \cdot \top = 0 = \top \cdot y \cdot \top$, so according to the Tarski rule $y = 0$ holds, too. Because the claim is symmetric in x and y the first equivalence is shown.
The second equivalence follows from the fact that 0 is the least element. ■

The next lemma gives some useful consequences of the Tarski rule:

Lemma 11.4.5 *Let $S = (M, +, 0, \cdot, 1, |\cdot\rangle, \langle\cdot|)$ be a modal semiring satisfying the Tarski rule. Then the following holds for all $x, y \in M$:*

1. $\top \cdot x \cdot \top \cdot y \cdot \top = 0 \;\Leftrightarrow\; (x \leq 0 \vee y \leq 0)$

2. $\top \cdot x \cdot \top \cdot y \cdot \top = 0 \;\Leftrightarrow\; \top \cdot y \cdot \top \cdot x \cdot \top = 0$

3. $\top \cdot x \cdot \top \cdot y \cdot \top = \top \cdot y \cdot \top \cdot x \cdot \top$

Proof: 1: If $x \leq 0 \vee y \leq 0$ we have clearly $\top \cdot x \cdot \top \cdot y \cdot \top = 0$ due to the annihilator properties of 0. Conversely, if $x \neq 0 \wedge y \neq 0$ we have $\top \cdot x \cdot \top \cdot y \cdot \top = \top \cdot y \cdot \top = \top \neq 0$.
2: This is a simple consequence of Part 1.
3: This follows from Part 2 and the Tarski rule since the only possible values of both $\top \cdot x \cdot \top \cdot y \cdot \top$ and $\top \cdot y \cdot \top \cdot x \cdot \top$ are 0 and $\top$. ■

From now till the end of this section we assume that every semiring under consideration fulfills the Tarski rule. In this case we can give equivalent formulations of symmetry:

Lemma 11.4.6 *The following statements are equivalent for arbitrary $x \in M$:*

1. $\forall\, p, q \in \mathsf{test}(S) : \top \cdot p \cdot x \cdot q \cdot \top = \top \cdot q \cdot x \cdot p \cdot \top$.

2. $\forall\, p, q \in \mathsf{test}(S) : p \cdot x \cdot q \leq 0 \Leftrightarrow q \cdot x \cdot p \leq 0$.

3. *x is symmetric.*

Proof: The equivalence of parts 1 and 2 is a simple consequence of Lemma 11.4.4. To show the equivalence of parts 2 and 3 we argue for arbitrary $x \in M$ as follows:

$(\forall p, q \in \mathsf{test}(S) \,:\, p \cdot x \cdot q \leq 0 \Leftrightarrow q \cdot x \cdot p \leq 0) \;\Leftrightarrow$
  { Part 1 of Definition 11.4.1 }
$(\forall p, q \in \mathsf{test}(S) \,:\, |x\rangle q \leq \neg p \Leftrightarrow \langle x| q \leq \neg p) \;\Leftrightarrow$
  { substitution $p \mapsto \neg p$, bijectivity of negation }
$(\forall p, q \in \mathsf{test}(S) \,:\, |x\rangle q \leq p \Leftrightarrow \langle x| q \leq p) \;\Leftrightarrow$
  { indirect equality }
$\forall q \in \mathsf{test}(S) \,:\, |x\rangle q = \langle x| q \;\Leftrightarrow$
  { Definition 11.4.3 }
x is symmetric. ■

This lemma implies the following one:

Lemma 11.4.7 *Let s be an arbitrary symmetric element $s \in M$ and p be an arbitrary test. Then $p \cdot s \cdot p$ is also symmetric.*

Proof: Assume arbitrary $q, q' \in \mathsf{test}(S)$. Then we can calculate:

$\top \cdot q \cdot (p \cdot s \cdot p) \cdot q' \cdot \top =$
{ associativity of $\cdot$ }
$\top \cdot (q \cdot p) \cdot s \cdot (p \cdot q') \cdot \top =$
{ symmetry of s, Part 1 of Lemma 11.4.6 }
$\top \cdot (p \cdot q') \cdot s \cdot (p \cdot q) \cdot \top =$
{ commutativity of $\cdot$ on $\mathsf{test}(S)$ }
$\top \cdot (q' \cdot p) \cdot s \cdot (p \cdot q) \cdot \top =$
{ associativity of $\cdot$ }
$\top \cdot q' \cdot (p \cdot s \cdot p) \cdot q \cdot \top$

This shows the claim due to Part 1 of Lemma 11.4.6. ■

An immediate consequence is the following corollary:

Corollary 11.4.1 *For every $p \in \mathsf{test}(S)$ the product $p \cdot \top \cdot p$ is symmetric; in particular, $\top$ is symmetric.*

Proof: Part 3 of Lemma 11.4.5 and Part 11.4.6 of Lemma 11.4.6 yield the symmetry of $\top$. Then $p \cdot \top \cdot p$ is symmetric due to Lemma 11.4.7. ■

Remark: In [Möl13] the concept of *m-symmetry* is introduced. In our terms, an element a is called *m-symmetric* if $\langle a| \leq |a\rangle$ holds. Then it is shown that for every m-symmetric element a and every test p the element $p \cdot a \cdot p$ is m-symmetric, too. We will show that the terms m-symmetric from [Möl13] and symmetric are equivalent.

Clearly, symmetry implies m-symmetry due to reflexivity of $\leq$. Moreover, due to antisymmetry of $\leq$, it suffices to show that $\langle a| \leq |a\rangle$ implies $|a\rangle \leq \langle a|$. So we fix an arbitrary test p and reason as follows:

$\langle a|p \leq |a\rangle p \Rightarrow$
{ Part 3 of Lemma 11.4.2 }
$p \leq |a]|a\rangle p \Rightarrow$
{ antitony of negation }
$\neg|a]|a\rangle p \leq \neg p \Rightarrow$
{ definition of forward box }
$|a\rangle\neg|a\rangle p \leq \neg p \Rightarrow$
{ definition of forward box }
$|a\rangle|a]\neg p \leq \neg p$

Because negation is a bijection on $\mathsf{test}(S)$ the last inequality implies $|a\rangle|a]q \leq q$ for every $q \in \mathsf{test}(S)$ and hence $(|a\rangle|a])\langle a|p \leq \langle a|p$ (note that $\langle a|p \in \mathsf{test}(S)$ holds). By associativity of function composition we obtain $|a\rangle(|a]\langle a|p) \leq \langle a|p$, and Part 2 of

Lemma 11.4.2 yields $|a\rangle p \leq \langle a|p$ due to isotony of diamond in the second argument (see Part 6 of Lemma 11.4.1). □

The last lemma in this subsection concerns the interplay between the greatest element and the diamond operators:

Lemma 11.4.8 *For arbitrary $p \in \mathsf{test}(S)$ with $p \neq 0$ we have $|\top\rangle p = 1 = \langle\top|p$. In particular, $|\top\rangle 1 = 1 = \langle\top|1$ holds.*

Proof: We only show $|\top\rangle 1 = 1$ and $|\top\rangle p = 1$; the equalities for the backward diamond are shown symmetrically. First we have $|\top\rangle 1 \geq |1\rangle 1$ due to isotony of the forward diamond. According to Part 1 of Lemma 11.4.1 we have $|1\rangle 1 = 1$. Since 1 is the greatest element in $\mathsf{test}(S)$ this implies $|\top\rangle 1 = 1$.
For the second equality we calculate:

$|\top\rangle p =$
  { $p = p \cdot 1$, Part 1 of Lemma 11.4.1 }
$|\top\rangle |p\rangle 1 =$
  { $|\top\rangle 1 = 1$ }
$|\top\rangle |p\rangle |\top\rangle 1 =$
  { Part 2 of Definition 11.4.1 }
$|\top \cdot p \cdot \top\rangle 1 =$
  { Tarski rule, $p \neq 0$ }
$|\top\rangle 1$
  { $|\top\rangle 1 = 1$ }
1,

and the proof is finished. ■

11.5 Equivalences

11.5.1 Equivalences and Fixpoints

Equivalence relation are reflexive, transitive and symmetric relations. In the last section we defined symmetry in the semiring framework, so the next step is to attack transitivity and reflexivity.

Reflexivity of a relation $R \subseteq X \times X$ can be characterised by $\mathsf{id}_X \subseteq R$, and transitivity by $R; R \subseteq R$. Since the identity corresponds to 1 and composition to multiplication in semirings, these characterisations can be translated as follows:

Definition 11.5.1 An element $x \in M$ is called *reflexive* if $1 \leq x$ and *transitive* if $x \cdot x \leq x$. A reflexive and transitive element is called a *preorder* and a symmetric preorder an *equivalence*.

A more liberal formulation of reflexivity and transitivity of x would be $|1\rangle \leq |x\rangle$ and $|x\rangle|x\rangle \leq |x\rangle$, resp., or in an equivalent way, $\langle 1| \leq \langle x|$ and $\langle x|\langle x| \leq \langle x|$. Clearly, Definition 11.5.1 implies the above conditions by isotony of the diamond operators and Part 2 of Definition 11.4.1, so we preferred a stronger version in our work.

The two formulations are indeed not equivalent. To see this, we consider the semiring of sets of walks in a graph with union as addition and pointwise lifted concatenation as multiplication (this means, given two sets of walks W_1 and W_2 we define the product by $W_1 \bowtie W_2 = \{w_1 \bowtie w_2 \,|\, w_1 \in W_1, w_2 \in W_2\}$). 0 in this semiring corresponds to the empty set (as in every semiring with union as addition) and 1 to the set of all walks of edge length 1. So the expression $1 \leq x$ means that x contains all these walks. On the other hand, $|1\rangle p \leq |x\rangle p$ means that x has to contain a walk from every node in p to some other node in p, but this walk need not to be of edge length 1.

For an equivalence relation R, the image operation of R is a closure operations, i.e., it is idempotent ($NR = (NR)R$), extensive ($N \subseteq NR$) and isotone ($N_1 \subseteq N_2 \Rightarrow N_1R \subseteq N_2R$). The same holds also for the preimage operator. Of course, idempotency implies that NR is a fixpoint of the function $\lambda N.NR$. These properties and one of its consequences are stated in the next lemma:

Lemma 11.5.1 *Let x be a preorder.*

1. *$|x\rangle$ and $\langle x|$ are closure operators.*
2. *If p is a test then $|x\rangle p$ is a fixpoint of $|x\rangle$ and $\langle x|p$ is a fixpoint of $\langle x|$.*
3. *The sets of fixpoints of $|x\rangle$ and $\langle x|$ each are closed under composition $\cdot$.*

Proof: 1: We have to show $|x\rangle p = |x\rangle|x\rangle p$ (idempotency), $p \leq |x\rangle p$ (extensitivity) and $q \leq p \Rightarrow |x\rangle q \leq |x\rangle p$ (isotony) for all tests q and p.
Isotony holds because all diamond operations are isotone. Because x is reflexive we have $1 \leq x$ and hence together with the isotony of the diamond the extensitivity. In order to show idempotency we calculate for an arbitrary test p as follows:

$|x\rangle|x\rangle p =$
  { Part 2 of Definition 11.4.1 }
$|x \cdot x\rangle p \leq$
  { transitivity of x, isotony of diamond }
$|x\rangle p =$
  { $p = 1 \cdot p = |1\rangle p$ (Part 1 of Lemma 11.4.1) }

$|x\rangle|1\rangle p \leq$
{ reflexivity of x, transitivity of diamond }
$|x\rangle|x\rangle p$

So this shows $|x\rangle|x\rangle p \leq |x\rangle p \leq |x\rangle|x\rangle p$, which implies the claim. The case of the backward diamond is treated symmetrically.

2: This follows simply from the idempotency of $|x\rangle$ and $\langle x|$, resp.

3: In [Bir67] it is shown in a more general setting that the two previous claims imply this one. ■

Later we will investigate the connection between fixpoints of an equivalence and equivalence classes of an equivalence relation. As known, the set of equivalence classes forms a set partition, so the complement of the union of equivalence classes can be written itself as the union of suitable equivalence classes. This is the quintessence of the following lemma:

Lemma 11.5.2 *Let r be an equivalence and p a fixpoint of the function $\langle r|$. Then $\neg p$ is a fixpoint of $\langle r|$, too (In the concrete setting of relations a similar result can be found in [SS93], p. 33.). The analogous claim holds for $|r\rangle$.*

Proof: Because r is reflexive and the diamonds are isotone we have the inequality $\langle r|\neg p \geq \neg p$. For the inverse inequality we can reason as follows:

$\langle r|\neg p \leq \neg p \Leftrightarrow$
{ Part 1 of Definition 11.4.1 }
$\neg p \cdot r \cdot p \leq 0 \Leftrightarrow$
{ Part 1 of Definition 11.4.1, again }
$|r\rangle p \leq p \Leftrightarrow$
{ symmetry of r }
$\langle r|p \leq p \Leftrightarrow$
{ assumption }
true

The claim for the forward diamond is shown analogously. ■

11.5.2 Equivalences and Partitions

In the previous Subsection 11.5.1 we enlightened the properties of fixpoints of equivalences. Equivalence classes are fixpoints of an equivalence relations, but not the only ones. Their special property in contrast to other fixpoints is that they are atomic in the lattice of all fixpoints of an equivalence relation. We will now show analogous properties in our framework.

Lemma 11.5.3 *Let r be an equivalence and F_r the set of all fixpoints of $|r\rangle$. Assume that F_r is atomic and denote the set of all its atoms by $A_r \subseteq F_r$. Then A_r is a partition.*

Proof: First we have to show the equality $\sum A_r = 1$. Because of $A_r \subseteq \mathsf{test}(S)$ we have $\sum A_r \leq 1$, so let us for the sake of contradiction assume that $\sum A_r < 1$. Then $\neg \sum A_r$ is different from zero, and due to Lemma 11.5.2 $\neg \sum A_r$ is also a fixpoint of r. So there has to be an atomic fixpoint a_r of r with $a_r \leq \neg \sum A_r$. Due to definition of A_r we have $a_r \in A_r$ and hence $a_r \leq \sum A_r$. This yields $a_r \cdot a_r \leq (\neg \sum A_r) \cdot (\sum A_r)$, which implies $a_r = 0$. But this is a contradiction to the fact that a_r is atomic.
In order to show the second property of a partition chose arbitrary $p, q \in A_r$ with $p \neq q$. Now by Lemma 11.5.1 it follows that $p \cdot q$ is again a fixpoint of r, and due to isotony of multiplication we have both $p \cdot q \leq p$ and $p \cdot q \leq q$. Because p and q are two different atomic fixpoints $p \cdot q = 0$ has to hold. ■

According to the introductory words of this subsection we can now make the following definition:

Definition 11.5.2 Let r be an equivalence. Then we call the set of atomic fixpoints of the function $\langle r|$, denoted by $Pa(r)$, the *equivalence classes* of r. Due to symmetry of r we can replace $\langle r|$ by $|r\rangle$.

A set partition corresponds in an obvious way to an equivalence relation. This fact is captured by the next definition encapsulated in a lemma:

Lemma 11.5.4 *Let P be a partition. Then $Eq(P) =_{df} \sum_{p \in P} p \cdot \top \cdot p$ is an equivalence. It is called the* equivalence induced by P.

Proof: First we show transitivity of $Eq(P)$. Therefore we calculate:

$$
\begin{aligned}
&(\sum_{p \in P} p \cdot \top \cdot p) \cdot (\sum_{p \in P} p \cdot \top \cdot p) = \\
&\quad \{ \text{ distributivity, associativity } \} \\
&\sum_{p,p' \in P} p \cdot \top \cdot p \cdot p' \cdot \top \cdot p' = \\
&\quad \{ \ \forall p, q \in P : p \cdot q = 0 \Leftrightarrow p \neq q \ \} \\
&\sum_{p \in P} p \cdot \top \cdot p \cdot p \cdot \top \cdot p = \\
&\quad \{ \text{ idempotency of multiplication on tests, associativity } \} \\
&\sum_{p \in P} p \cdot (\top \cdot p \cdot \top) \cdot p = \\
&\quad \{ \text{ Tarski rule } \} \\
&\sum_{p \in P} p \cdot \top \cdot p
\end{aligned}
$$

This shows even the sharper equality $(\sum_{p\in P} p\cdot\top\cdot p)\cdot(\sum_{p\in P} p\cdot\top\cdot p) = \sum_{p\in P} p\cdot\top\cdot p$ instead of the sufficient inequality.

Reflexivity is shown by the following calculation:

$$\begin{array}{l}
\sum_{p\in P} p\cdot\top\cdot p \geq \\
\quad \{\ \top \geq 1,\ \text{isotony of multiplication}\ \} \\
\sum_{p\in P} p\cdot 1\cdot p = \\
\quad \{\ \text{idempotency of multiplication on tests}\ \} \\
\sum_{p\in P} p = \\
\quad \{\ \text{partition properties}\ \} \\
1
\end{array}$$

To show symmetry we chose an arbitrary $q \in \mathsf{test}(S)$ and calculate:

$$\begin{array}{l}
|\sum_{p\in P} p\cdot\top\cdot p\rangle q = \\
\quad \{\ \text{distributivity of summation over diamonds}\ \} \\
\sum_{p\in P} |p\cdot\top\cdot p\rangle q = \\
\quad \{\ \text{Corollary 11.4.1}\ \} \\
\sum_{p\in P} \langle p\cdot\top\cdot p| q = \\
\quad \{\ \text{distributivity of summation over diamonds}\ \} \\
\langle \sum_{p\in P} p\cdot\top\cdot p| q
\end{array}$$

So we have shown $|\sum_{p\in P} p\cdot\top\cdot p\rangle = \langle \sum_{p\in P} p\cdot\top\cdot p|$, as desired. ■

An equivalence relation relates exactly the elements inside its equivalence classes. This is expressed in the following lemma:

Lemma 11.5.5 *Let r be an equivalence and $p, q \in Pa(r)$ arbitrary. Then the equivalence $p\cdot r\cdot q = 0 \Leftrightarrow p \neq q$ holds.*

Proof: According to Part 11.4.1 of Definition 11.4.1 the expression $p\cdot r\cdot q = 0$ is equivalent to $\langle r|p \leq \neg q$. Because p is a fixpoint of r this holds iff $p \leq \neg q$. Applying the shunting rule yields the equivalent statement $p\cdot q = 0$. Now by Lemma 11.5.3 we know that $Pa(r)$ is a partition, so the last statement is equivalent to $p \neq q$. ■

A similar property is stated by the next lemma:

Lemma 11.5.6 *Let r be an equivalence. Then r respects $Pa(r)$.*

Proof: We have to show $r = \sum_{p \in Pa(r)} p \cdot r \cdot p$. This is done as follows:

$$
\begin{array}{l}
r = \\
\quad \{\ r = 1 \cdot r \cdot 1, \text{ partition properties }\ \} \\
(\sum Pa(r)) \cdot r \cdot (\sum Pa(r)) = \\
\quad \{\ \text{distributivity }\ \} \\
\quad \sum_{p,p' \in Pa(r)} p \cdot r \cdot p' = \\
\quad \{\ \text{splitting the sum }\ \} \\
\quad \sum_{p \in Pa(r)} p \cdot r \cdot p + \sum_{p,p' \in Pa(r), p \neq p'} p \cdot r \cdot p' = \\
\quad \{\ \text{Lemma 11.5.5 }\ \} \\
\quad \sum_{p \in Pa(r)} p \cdot r \cdot p
\end{array}
$$

■

The next lemma is mainly of technical interest:

Lemma 11.5.7 *For a partition P and arbitrary test q we have the equality $|Eq(P)\rangle q = \sum_{p \in P \wedge p \cdot q \neq 0} p$.*

Proof: The proof is done by the following calculation:

$$
\begin{array}{l}
|Eq(P)\rangle q = \\
\quad \{\ \text{definition of } Eq(P)\ \} \\
|\sum_{p \in P} p \cdot \top \cdot p\rangle q = \\
\quad \{\ \text{distributivity of sum over diamond }\ \} \\
\sum_{p \in P} |p \cdot \top \cdot p\rangle q = \\
\quad \{\ \text{Part 2 of Definition 11.4.1 }\ \} \\
\sum_{p \in P} |p\rangle|\top\rangle|p\rangle q = \\
\quad \{\ \text{Part 1 of Lemma 11.4.1 }\ \} \\
\sum_{p \in P} |p\rangle|\top\rangle(p \cdot q) = \\
\quad \{\ \text{Part 2 of Lemma 11.4.1 }\ \} \\
\quad \sum_{p \in P \wedge p \cdot q \neq 0} |p\rangle|\top\rangle(p \cdot q) = \\
\quad \{\ \text{Lemma 11.4.8 }\ \} \\
\quad \sum_{p \in P \wedge p \cdot q \neq 0} |p\rangle 1 = \\
\quad \{\ p = 1 \cdot p = |1\rangle p \text{ (Part 1 of Lemma 11.4.1) }\ \} \\
\quad \sum_{p \in P \wedge p \cdot q \neq 0} p
\end{array}
$$

■

Now we can prove the main theorem of this subsection, which connects the operations Eq and Pa:

Theorem 11.5.1 *Let r be an equivalence and P a partition. Then the following holds:*

1. $r \leq Eq(Pa(r))$.
2. $P = Pa(Eq(P))$.

This means in particular that Pa and Eq form a Galois connection (cf. [EKMS94]).

Proof: 1: This is done by calculation:

$$
\begin{array}{l}
r = \\
\quad \{ \text{ Lemma 11.5.6 } \} \\
\quad \sum\limits_{p \in Pa(r)} p \cdot r \cdot p \leq \\
\quad \{ \text{ isotony of multiplication } \} \\
\quad \sum\limits_{p \in Pa(r)} p \cdot \top \cdot p = \\
\quad \{ \text{ Lemma 11.5.4 } \} \\
Eq(Pa(r))
\end{array}
$$

2: Because P is a partition we know due to Lemma 11.5.7 that every $p \in P$ is a fixpoint of $|Eq(P)\rangle$. We will now show that the elements of P are even atomic fixpoints of $|Eq(P)\rangle$. Therefore we fix an arbitrary $p \in P$ and consider an arbitrary test $q \neq 0$ with $q < p$. Then we have $p \cdot q = q \neq 0$ and $p' \cdot q \leq p' \cdot p = 0$ for all $p' \in P$ with $p' \neq p$. So again by Lemma 11.5.7 we get $|Eq(P)\rangle q = p \neq q$, which means that q is no fixpoint of $|Eq(P)\rangle$. By means of Definition 11.5.2 we have till now $P \subseteq Pa(Eq(P))$.

Now it remains to show that every atomic fixpoint of $|Eq(P)\rangle$ is also an element of P. Therefore consider first an arbitrary fixpoint q of $|Eq(P)\rangle$. Then we have $q = |Eq(P)\rangle q = \sum\limits_{p \in P \wedge p \cdot q \neq 0} p$ due to the fixpoint property of q and Lemma 11.5.7. This sum decomposition holds for all fixpoints of $|Eq(P)\rangle$, in particular for every atomic fixpoint. Due to atomicity an atomic fixpoint of $|Eq(P)\rangle$ can not be written as the sum of different atomic fixpoints of $|Eq(P)\rangle$, i.e. elements of $Pa(Eq(P))$. Hence the sum $\sum\limits_{p \in P \wedge p \cdot q \neq 0} p$ has to contain exactly one summand unequal to 0, which has to arise in the case $p = q$. So q is also an element of P which implies $|Eq(P)\rangle \subseteq P$.

The fact that Pa and Eq form a Galois connection form a Galois connection follows from these properties together with the isotony of Pa and Eq by standard results (cf. again [EKMS94]). ■

If we consider the semiring of relations the inequality of Part1 is even an equality. However, in general semiring a strict inequality can hold. Therefore consider again the semiring of paths over the graph $G = (\{x\}, \{(x, x)\})$. In this semiring the element $1 = \{x\}$ is an equivalence (as in every modal semiring), and the only partition is given by $P = \{1\} = \{\{x\}\}$. But then we have $Eq(Pa(1)) = \top \neq 1$ because here $\top$ is the union of all finite paths of the form $xxx\ldots$.

11.6 Atomic Tests and Equivalence Classes

In the previous subsection we investigated the relationship between partitions and atomic fixpoints of certain functions. Now we will see how atomic tests fit into this framework. In the relation semiring, atomic tests correspond to singleton subsets, i.e. to elements of the carrier set. So the next lemma can be interpreted as 'Two elements are connected by an equivalence relation iff they are in the same equivalence class.'

Lemma 11.6.1 *Let r be an equivalence and p, q be atomic tests. Then we have the equivalence $p \cdot r \cdot q \neq 0 \Leftrightarrow |r\rangle p = |r\rangle q$.*

Proof: '$\Rightarrow$': First we have:

$p \cdot r \cdot q \neq 0 \Rightarrow$
{ Lemma 11.4.3 }
$p \leq |r\rangle q \Rightarrow$
{ isotony of the diamond }
$|r\rangle p \leq |r\rangle|r\rangle q \Rightarrow$
{ Part 2 of Definition 11.4.1 }
$|r\rangle p \leq |r \cdot r\rangle q \Rightarrow$
{ transitivity of r, isotony of the diamond }
$|r\rangle p \leq |r\rangle q$.

Symmetrically we can show $\langle r|q \leq \langle r|p$. Because r is symmetric this is equivalent to $|r\rangle q \leq |r\rangle p$. Now the equality follows from the two inequalities.

'$\Leftarrow$': We have $p = 1 \cdot p = |1\rangle p \leq |r\rangle p$ due to Lemma 11.4.1, reflexivity of r and isotony of the diamond. So by assumption we have $p \leq |r\rangle q$, and the claim follows from Lemma 11.4.3. ■

An equivalence relation respects a given set partition iff the set of its equivalence classes is a refinement of the given set partition. This property holds also in our general setting, as stated by the following theorem:

Theorem 11.6.1 *Let r be an equivalence. Then r respects a partition P iff $Pa(r)$ refines P.*

Proof: '$\Rightarrow$': Let us assume that $Pa(r)$ is no refinement of P. Then by Lemma 11.3.5 there is a $p \in P$ and $q, q' \in Pa(r)$ with $q \neq q'$, $p \cdot q \neq 0$ and $p \cdot q' \neq 0$. Because we assumed that $\mathsf{test}(S)$ is atomic there are two atomic tests $p_a, p'_a \in \mathsf{atest}(S)$ such that $p_a \leq p \cdot q$ and $p'_a \leq p \cdot q'$ hold. In particular, this implies $p_a \leq p$ and $p'_a \leq p$, so the equivalence classes of p_a and p'_a under r equal both p Now Lemma 11.6.1 yields $p_a \cdot r \cdot p'_a \neq 0$, and by isotony of multiplication we obtain $q \cdot r \cdot q' \neq 0$. But now r can not respect P due to Lemma 11.3.7.

'$\Leftarrow$': By Lemma 11.5.6 we know that r has to respect $Pa(r)$. Then r respects also P due to Theorem 11.3.1. ■

In the concrete setting of relations equivalence classes are defined as the image (or equivalently preimage) of single elements of an equivalence relation. Here we defined them via atomic fixpoints. A very aesthetic result is that our definition is equivalent to this one in the context of semirings:

Theorem 11.6.2 *Let r be an equivalence and p an atomic test. Then $|r\rangle p$ is an equivalence class of r. It is called the* equivalence class of p under r *and is denoted by $[p]_r$.*

Proof: To satisfy Definition 11.5.2 we have to show that $|r\rangle p$ is atomic in the set of fixpoints of $|r\rangle$. So we consider an arbitrary test q with $0 \neq |r\rangle q \leq |r\rangle p$. Because of Part 1 of Lemma 11.4.1 we can conclude that $q \neq 0$ holds. The set $\mathsf{test}(S)$ is atomic, so there is a nonempty set $Q' \subseteq \mathsf{atest}(S)$ with $q = \sum Q'$. Due to distributivity of the diamond (cf. the remark after Lemma 11.4.1) we can write the inequality $|r\rangle q \leq |r\rangle p$ in the equivalent form $\forall q' \in Q' : |r\rangle q' \leq |r\rangle p$. By Part 1 of Lemma 11.4.1 we have $|r\rangle q' = r \cdot q'$ and reflexivity of r yields $r \cdot q' \leq q'$, altogether this implies $\forall q' \in Q' : q' \leq |r\rangle p$. Now we have $\forall q' \in Q' : q' \cdot r \cdot p \neq 0$ due to Lemma 11.4.3 and by Lemma 11.6.1 we obtain $\forall q' \in Q' : |r\rangle q' = |r\rangle p$. But this implies the desired equality $|r\rangle q = \sum_{q' \in Q'} |r\rangle q' = |r\rangle p$. ■

11.7 Bisimulations

In Section 4 we mentioned immediately after Definition 4.2.1 an alternative characterisation of bisimulations. A relation B is a bisimulation between two set labelled graphs $G_1 = ((V_1, E_1), g_1)$ and $G_2 = ((V_2, E_2), g_2)$ if $B^\circ; \overset{l}{\rightarrow}_{g_1} \subseteq \overset{l}{\rightarrow}_{g_2}; B^\circ$ and $B; \overset{l}{\rightarrow}_{g_2} \subseteq \overset{l}{\rightarrow}_{g_1}; B$ hold for all $l \in L$. In Section 11.4 we showed how we can deal with the converse in semirings, so we can translate this into the following definition:

Definition 11.7.1 An element $b \in M$ is called a *bisimulation* for $g \in M$ if $|b\rangle|g\rangle \leq |g\rangle|b\rangle \wedge \langle b||g\rangle \leq |g\rangle\langle b|$ holds. For an element $g \in M$ the set of all bisimulations for g

is denoted by bisim_g.

Note that by this definition 0 is a bisimulation for every $g \in M$. We will deal with this later. Moreover, the term bisimulation from the previous definition is equivalent to the term 'autobisimulation' from Section 4.

As already mentioned in Section 4 the set of bisimulations between two set labelled graphs is closed under union and relational composition. The analogous fact in the framework of semirings is stated in the following lemma:

Lemma 11.7.1 *Let $g \in M$ and $b, b' \in \mathsf{bisim}_g$ be arbitrary. Then $b + b' \in \mathsf{bisim}_g$ and $b \cdot b' \in \mathsf{bisim}_g$ hold. In the case of a dioid we have even the implication $B \subseteq \mathsf{bisim}_g \Rightarrow \sum_{b \in B} b \subseteq \mathsf{bisim}_g$.*

Proof: The claims for the sum follow directly from Part 1 of Lemma 11.4.1 and the afterward remark. The claim for the product is a simple consequence of Part 2 of Definition 11.4.1. ■

Till now we avoided the explicit use of a converse operation via the use of the backward and forward diamond. We will now not introduce a new operation because it turns out that the existence of a pseudoconverse is sufficient for our purposes.

Definition 11.7.2 For an element $x \in M$ we call an element $y \in M$ a *pseudoconverse* of x if $|x\rangle = \langle y|$; in this case we use the abbreviation $\mathsf{pscon}(x, y)$.

Clearly, a symmetric element is a pseudoconverse of itself. From now on we require that for every element a (not necessarily unique) pseudoconverse exists. In the concrete semiring of relations the pseudoconverse is unique and coincides with the common converse of a relation.

Intuitively, we would expect that the relation pscon is symmetric. In this expectation we will not be disappointed by the semirings:

Lemma 11.7.2 *For arbitrary $x, y \in M$ we have $\mathsf{pscon}(x, y) \Leftrightarrow \mathsf{pscon}(y, x)$.*

Proof: We show only the inequality $\langle x| \leq |y\rangle$ (the reverse inequality can be shown symmetrically, and the other direction of the equivalence follows by exchanging the roles of x and y). So we pick an arbitrary $p \in \mathsf{test}(S)$ and reason as follows:

$\langle x|p \leq |y\rangle p \Leftrightarrow$
  { Part 1 of Definition 11.4.1 }
$p \cdot x \cdot (\neg|y\rangle p) \leq 0 \Leftrightarrow$

$$
\begin{array}{l}
\quad \{ \text{ Part 1 of Definition 11.4.1 } \} \\
|x\rangle(\neg|y\rangle p) \leq \neg p \Leftarrow \\
\quad \{ \ |x\rangle \leq \langle y|, \text{ isotony of the diamond } \} \\
\langle y|(\neg|y\rangle p) \leq \neg p \Leftrightarrow \\
\quad \{ \text{ Part 1 of Definition 11.4.1 } \} \\
(\neg|y\rangle p) \cdot y \cdot p \leq 0 \Leftrightarrow \\
\quad \{ \text{ Part 1 of Definition 11.4.1 } \} \\
|y\rangle p \leq |y\rangle p \Leftrightarrow \\
\quad \{ \text{ reflexivity of } \leq \} \\
\mathsf{true}
\end{array}
$$

■

The union of a relation with its converse is a symmetric relation. In our setting this is expressed by the following lemma:

Lemma 11.7.3 *Let $x, y \in M$ be arbitrary with $\mathsf{pscon}(x, y)$. Then $x+y$ is symmetric.*

Proof: Let x and y be as above and let $p \in \mathsf{test}(S)$ be arbitrary test. Then we can calculate as follows:

$$
\begin{array}{l}
\langle x + y|p = \\
\quad \{ \text{ Part 4 of Lemma 11.4.1 } \} \\
\langle x|p + \langle y|p = \\
\quad \{ \ \mathsf{pscon}(x, y), \text{ Lemma 11.7.2 } \} \\
|y\rangle p + |x\rangle p = \\
\quad \{ \text{ Part 4 of Lemma 11.4.1, commutativity of } + \ \} \\
|x + y\rangle p
\end{array}
$$

So $x + y$ is symmetric in the sense of Definition 11.4.3. ■

Autobisimulations relate states with equivalent behaviour, so the converse of an autobisimulation is also an autobisimulation. The next lemma deals with this fact:

Lemma 11.7.4 *Let $g, x, y \in M$ be arbitrary with $x \in \mathsf{bisim}_g$ and $\mathsf{pscon}(x, y)$. Then we have $y \in \mathsf{bisim}_g$.*

Proof: This is an easy consequence from the definitions of bisimulation (Definition 11.7.1) and pseudoconverse (Definition 11.7.2). ■

Due to Lemma 11.7.1 the set of bisimulations bisim_g for arbitrary $g \in M$ is in a dioid closed under arbitrary sums. Hence we have $\sum_{b \in \mathsf{bisim}_g} b \in \mathsf{bisim}_g$ and, moreover, $b' \leq \sum_{b \in \mathsf{bisim}_g} b$ for all $b' \in \mathsf{bisim}_g$. So we define the *coarsest bisimulation* for g by

$\widehat{g} =_{df} \sum_{b \in \mathsf{bisim}_g} b$. Analogously to relations this is also here an equivalence:

Theorem 11.7.1 *For every $g \in M$ the coarsest bisimulation $\widehat{g}$ for g is an equivalence.*

Proof: Due to Definition 11.7.1 and the laws from Lemma 11.4.1 we have $1 \in \mathsf{bisim}_g$, hence $1 \leq \widehat{g}$; so $\widehat{g}$ is reflexive.
To show transitivity we first observe that $\{b \cdot b' \mid b, b' \in \mathsf{bisim}_g\} \subseteq \mathsf{bisim}_g$ holds due to Lemma 11.7.1. Then we can calculate:

$$
\begin{array}{l}
\widehat{g} \cdot \widehat{g} = \\
\quad \{ \text{ definition of } \widehat{g} \ \} \\
(\sum_{b \in \mathsf{bisim}_g} b) \cdot (\sum_{b' \in \mathsf{bisim}_g} b') = \\
\quad \{ \text{ Distributivity } \} \\
\quad \sum_{b, b' \in \mathsf{bisim}_g} b \cdot b' \leq \\
\quad \{ \ \{b \cdot b' \mid b, b' \in \mathsf{bisim}_g\} \subseteq \mathsf{bisim}_g \ \} \\
\quad \sum_{b \in \mathsf{bisim}_g} b = \\
\quad \{ \text{ definition of } \widehat{g} \ \} \\
\widehat{g}
\end{array}
$$

To show symmetry we use the fact that due to Lemma 11.7.4 for every $b \in \mathsf{bisim}_g$ there is a $b' \in \mathsf{bisim}_g$ with $\mathsf{pscon}(b, b')$ (recall the assumption after Definition 11.7.2). So due to idempotency of summation we have the equality $\sum_{b \in \mathsf{bisim}_g} b = \sum_{b \in \mathsf{bisim}_g} (b + b')$. Now due to Lemma 11.7.3 the second sum contains only symmetric summands, and is hence symmetric itself (see the remark after Definition 11.4.3). ■

Since $\widehat{g}$ is reflexive we have $0 \neq 1 \leq \widehat{g}$, so we have always a nonempty bisimulation at our effort.

Let us take a look at a equivalence class E of an autobisimulation B on a set labelled graph $G = ((V, E), g)$. If for an element $e \in E$ a transition $e \xrightarrow{l}_g f$ for some label l is possible, then for every other $e' \in E$ a transition $e' \xrightarrow{l}_g f'$ is possible where f and f' are in the same equivalence class of B. Clearly, an analogous observation also holds for subsets of equivalence classes. This stability property is the content of the following theorem:

Theorem 11.7.2 *Let $g \in M$ be arbitrary and $p, q \in \mathsf{atest}(S)$ be atomic tests. If $p \cdot g \cdot q \neq 0$ then for all $p' \leq [p]_{\widehat{g}}$ with $p' \neq 0$ we have $p' \cdot g \cdot [q]_{\widehat{g}} \neq 0$.*

Proof: We will first show the claim only for atomic p'. Now $\widehat{g}$ is an equivalence (Theorem 11.7.1) and because of $|\widehat{g}\rangle p' = [p]_{\widehat{g}} = |\widehat{g}\rangle [p]_{\widehat{g}}$ we can apply Lemma 11.6.1

and obtain $p' \cdot \widehat{g} \cdot p \neq 0$. By Lemma 11.4.3 this implies $p' \leq |\widehat{g}\rangle p$. Analogously, we obtain $p \leq |g\rangle q$ by Lemma 11.4.3 from the assumption $p \cdot g \cdot q \neq 0$ and atomicity of p. Now we can calculate:

$$
\begin{aligned}
&0 \neq \\
&\quad \{ \text{ assumption } \} \\
&p' \leq \\
&\quad \{ \text{ above argumentation } \} \\
&|\widehat{g}\rangle p \leq \\
&\quad \{ \text{ above argumentation, isotony of diamond } \} \\
&|\widehat{g}\rangle |g\rangle q \leq \\
&\quad \{ \widehat{g} \in \mathsf{bisim}_g, \text{ Definition 11.7.1 } \} \\
&|g\rangle |\widehat{g}\rangle q = \\
&\quad \{ \text{ symmetry of } \widehat{g}, \text{ cf. Theorem 11.7.1 and Definition 11.5.1 } \} \\
&|g\rangle \langle \widehat{g}| q = \\
&\quad \{ \text{ Definition 11.5.2 } \} \\
&|g\rangle [q]_{\widehat{g}}
\end{aligned}
$$

Now, by Lemma 11.4.3 we have the desired inequality $p' \cdot g \cdot [q]_{\widehat{g}} \neq 0$. ■

So, $Pa(\widehat{g})$ is the coarsest partition which is stable with respect to g.

11.8 Application to a Simple Control Objective

We will now sketch how the approach of quotient and expansion can be used to ensure a simple liveness property. Semirings are a suitable model in our example, because they represent an easily manageable description of unlabelled transition systems since they are in principle the same as a relation $R \subseteq M \times M$.

The property we will ensure is that every node with an ingoing edge has also an outgoing edge. Moreover, we are looking for a maximal relation with respect to $\subseteq$ fulfilling this property. First, we will define the above liveness property in our semiring setting. An equivalent relational description is that if the preimage of a node set is non-empty its image has also to be non-empty. The formulation $\forall M' \subseteq M : RM' \neq \emptyset \Rightarrow M'R \neq \emptyset$ in relation algebra translates in semirings into the following definition:

Definition 11.8.1 An element $g \in M$ is called *live* if the implication $|g\rangle p \neq 0 \Rightarrow \langle g|p \neq 0$ holds for all $p \in \mathsf{test}(S)$. An element $g' \in M$ is called a *live part* of an element g if g' is live and $g' \leq g$.

The equality $|0\rangle p = 0$ for all tests $p \in \mathsf{test}(S)$ implies that 0 is live and because of $0 \leq x$ for all $x \in M$ we know that 0 is a live part of every element $x \in M$. In the case

of a dioid the diamond operators distribute over arbitrary sums, so the sum of live elements is again a live element. So, similarly to bisimulations, the set of live parts of an element g is closed under arbitrary sum, hence for every $g \in M$ there is a greatest live part (namely the sum of all its live parts), which we will denote by glp_g.

The original definition of liveness of a relation $R \subseteq M \times M$ at the beginning of this subsection was based on single elements of the carrier set M. Few lines later we gave an equivalent relational algebraic characterisation of liveness without looking at single elements. This equivalence holds also in semirings (remember that nodes of a graph correspond to atomic tests):

Lemma 11.8.1 *An element $g \in M$ is live iff the implication $|g\rangle p \neq 0 \Rightarrow \langle g|p \neq 0$ holds for every atomic test $p \in \mathsf{atest}(S)$.*

Proof: The direction '$\Rightarrow$' is clear since $\mathsf{atest}(S) \subseteq \mathsf{test}(S)$ holds. So let $p' \in \mathsf{test}(S)$ be an arbitrary test. Then we can reason as follows:

$$
\begin{array}{l}
|g\rangle p' \neq 0 \Rightarrow \langle g|p' \neq 0 \;\Leftrightarrow \\
\quad \{ \text{ atomicity of } \mathsf{test}(S) \} \\
|g\rangle \sum\limits_{p'' \in \mathsf{atest}(S), p'' \leq p'} p'' \neq 0 \Rightarrow \langle g| \sum\limits_{p'' \in \mathsf{atest}(S), p'' \leq p'} p'' \neq 0 \;\Leftrightarrow \\
\quad \{ \text{ distributivity of diamond over sum } \} \\
\sum\limits_{p'' \in \mathsf{atest}(S), p'' \leq p'} |g\rangle p'' \neq 0 \Rightarrow \sum\limits_{p'' \in \mathsf{atest}(S), p'' \leq p'} \langle g|p'' \neq 0 \;\Leftrightarrow \\
\quad \{ \text{ sum-irreducibility of } 0 \} \\
(\exists p'' \in \{p'' \in \mathsf{atest}(S), p'' \leq p'\} : |g\rangle p'' \neq 0 \Rightarrow \\
\;\exists p'' \in \{p'' \in \mathsf{atest}(S), p'' \leq p'\} : \langle g|p'' \neq 0)
\end{array}
$$

But now by assumption the last implication evaluates to true. ■

A more technical property of the greatest live part is expressed in the following lemma:

Lemma 11.8.2 *Consider an arbitrary element g and its greatest live part glp_g. Then for all atomic tests $p, q \in \mathsf{atest}(S)$ we have $p \cdot \mathsf{glp}_g \cdot q = p \cdot g \cdot q \vee p \cdot \mathsf{glp}_g \cdot q = 0$.*

Proof: Let g, p and q be as above. Because of $\mathsf{glp}_g \leq g$ we have $p \cdot \mathsf{glp}_g \cdot q \leq p \cdot g \cdot q$. If $p \cdot \mathsf{glp}_g \cdot q = 0$ or $p \cdot \mathsf{glp}_g \cdot q = p \cdot g \cdot q$ hold there is nothing to show, so assume that $0 < p \cdot \mathsf{glp}_g \cdot q < p \cdot g \cdot q$ holds. But then the element $g' =_{df} g + p \cdot g \cdot q$ would also be a live part of g (this holds because of Lemma 11.8.1, Definition 11.4.1 and additivity of the diamond operators) with $\mathsf{glp}_g < g'$ which contradicts the definition of glp_g. ■

In the sequel we will use the concept of the *marker* $\delta_g(p, q)$ of an element $g \in M$ and two tests $p, q \in \mathsf{test}(S)$ (its purpose will become clear in Definition 11.8.3). It indicates whether g allows a transition from p to q. If so, it is a restriction of $\top$, otherwise it becomes 0. This is the content of the next definition:

Definition 11.8.2 For an element $g \in M$ the *marker* function $\delta_g(\cdot,\cdot) : \mathsf{test}(S) \times \mathsf{test}(S) \to M$ is defined by $\delta_g(p,q) = p \cdot \top \cdot q$ if $p \cdot g \cdot q \neq 0$, and is 0 otherwise.

The next problem we meet is how to express the quotient g/b of an arbitrary $g \in M$ and a bisimulation equivalence $b \in \mathsf{bisim}_g$ in a sense analogous to Definition 6.2.1. There the nodes of the quotient are equivalence classes of a bisimulation equivalence, and they are connected iff there is an edge between suitable elements of them. So an obvious idea would be to define $g/b =_{df} \sum_{p,q \in Pa(b)} \delta_g(p,q)$. Then we have $p \cdot (g/b) \cdot q \neq 0$ for all $p, q \in Pa(b)$ iff there are atomic tests $p' \leq p$ and $q' \in q$ with $p \cdot g \cdot q \neq 0$. In the later course we will find it more convenient to use a more abstract definition via systems of representatives:

Definition 11.8.3 Let r be an equivalence. A set $\mathsf{Rep} \subseteq \mathsf{atest}(S)$ of atomic tests is called a *system of representatives (SOR)* for r if $\sum_{p \in \mathsf{Rep}} [p]_r = 1$ and $p, q \in \mathsf{Rep} \wedge p \neq q \;\Rightarrow\; [p]_r \cdot [q]_r = 0$ hold. For an arbitrary element $g \in M$ and a bisimulation equivalence $b \in \mathsf{bisim}_g$ we call an element $h \in M$ a *quotient witness* of g and b if there is a SOR Rep of b with $h = \sum_{p,q \in \mathsf{Rep}} p \cdot \delta_g([p]_{\widehat{g}}, [q]_{\widehat{g}}) \cdot q$. In this case we call Rep the *associated* SOR of h and the elements $(p,q) \in \mathsf{Rep}^2$ with $p \cdot h \cdot q \neq 0$ its *edges*, which we denote by edges_h. The set of all quotient witnesses of g and b is denoted by $\mathsf{qw}(g,b)$.

Remark: If we use the notion Rep it will be clear from the context which equivalence is considered. □

So a system of representatives in the semiring of relations is the same as usual. A quotient witness in this case is a relation isomorphic to the quotient as in Definition 6.2.1. The only difference is that the nodes are not equivalence classes but representatives. Note that the edges in a quotient witness need not be edges in the original relation. The advantage of dealing with quotient witnesses instead of the above construction $\sum_{p,q \in Pa(b)} \delta_g(p,q)$ is that we have less edges to handle.

For the modelling of the refinement operation in the sense of Definition 5.2.1 we can use the order of the semiring, i.e. an element x *refines* an element y iff $x \leq y$ holds. The last operation we have to handle is the expansion operation as in Definition 6.3.1. This is done as follows:

Definition 11.8.4 Let h be a quotient witness for elements g and b, and let h' be a refinement of h. Then the *expansion* $h' \backslash b$ of h' by b is defined by $h' \backslash b =_{df} \sum_{(p,q) \in \mathsf{edges}_{h'}} [p]_b \cdot g \cdot [q]_b$.

This corresponds in an obvious way to Definition 6.3.1. Clearly, $h'\backslash b \leq g$ holds due to $[p]_b \leq 1$, $[q]_b \leq 1$, isotony of multiplication and the supremum property of the sum.

Now we put the pieces together and show that a greatest live part can be constructed from a greatest live part of a quotient witness:

Theorem 11.8.1 *Let h be a quotient witness for elements g and b. Then $\mathsf{glp}_h\backslash b = \mathsf{glp}_g$ holds.*

Proof: To ease the writing we set $g' =_{df} \mathsf{glp}_h\backslash b$ As remarked above we have $g' \leq g$ because glp_h is a refinement of h.

To show liveness of g' it suffices by Lemma 11.8.1 to show the implication $|g'\rangle p \neq 0 \Rightarrow \langle g'|p \neq 0$ for all atomic tests $p \in \mathsf{atest}(S)$. So we consider an arbitrary atomic test $p \in \mathsf{atest}(S)$ with the property $|g'\rangle p \neq 0$ and fix an arbitrary system of representatives Rep for b. The equivalence class $[p]_b$ has a representative in Rep, which we will denote by p_b. By construction of g' we have a pair $(p_b, p'_b) \in \mathsf{edges}_{\mathsf{glp}_h}$ (which means $p_b \cdot \mathsf{glp}_h \cdot p'_b \neq 0$). Due to the definition of the quotient we have $p_b \cdot [p'_b]_b \neq 0$. Because p and p_b are both contained in the same equivalence class $[p]_b$ (formally $[p]_b = [p_b]_b$) this yields $p \cdot g \cdot [p'_b]_b \neq 0$ and hence $p \cdot g' \cdot [p'_b]_b \neq 0$ by construction of g'. This implies $\langle g'|p \neq 0$, as desired, so g' is indeed a live part of g.

Assume now that $g' < \mathsf{glp}_g$ holds. Then there are atomic tests p, q with $p \cdot g' \cdot q < p \cdot \mathsf{glp}_g \cdot q$. By construction of g' and Lemma 11.8.2 this means $p \cdot g' \cdot q = 0$ and $p \cdot \mathsf{glp}_g \cdot q = p \cdot g \cdot q \neq 0$. Let us now consider the element $\hat{h} =_{df} \sum_{p',q' \in \mathsf{Rep}} p \cdot \delta_{\mathsf{glp}_q}([p']_b, [q']_b)$.

Clearly, $\hat{h} \leq h$ holds because of $\mathsf{glp}_g \leq g$. Choose now $p_b, q_b \in \mathsf{Rep}$ with $p \in [p_b]_b$ and $q[q_b]_b$. By construction of g', $(p_b, q_b) \notin \mathsf{edges}_{\mathsf{glp}_h}$, and by the above observation, $[p_b]_b \cdot \mathsf{glp}_g \cdot q_b$ hold. Let now $\hat{p} \in \mathsf{Rep}$ be arbitrary with $|\hat{h}\rangle\hat{p} \neq 0$. Then we have $|\mathsf{glp}_g\rangle[\hat{p}]_b \neq 0$ and hence because of liveness of glp_g also $\langle \mathsf{glp}_g|[\hat{p}]_b \neq 0$. But this means that $\hat{h}$ is a live part of h with $\mathsf{glp}_h < \hat{h}$ which contradicts the definition of glp_h. ■

We will now illustrate this construction on our running example relation (see Page 99). Its coarsest bisimulation is the equivalence $[0,2]^2 \cup [4,6]^2 \cup (]2,4[\cup]6,9[)^2$ with the three equivalence classes $[0,2]$, $[4,6]$ and $]2,4[\cup]6,9[$. A possible quotient witness is the relation $\{(1,4),(4,8),(4,1)\}$ with the greatest live part $\{(1,4),(4,1)\}$. The above construction yields the relation $\{(x,y) \in \mathbb{R}^2 \mid (x \in [0,2] \wedge y = x+4) \vee x \in [4,6] \wedge y = x-4\}$. which is indeed the greatest live part of our example relation.

Chapter 12

Automated Theorem Provers

In this chapter we earn the fruits of the work done in the previous chapter. We show that the algebraic characterisation we developed there can be used as input for automated theorem provers and evaluate this approach using Prover9 and Mace4. This chapter leaves the straight way a little bit and does not only deal with the main subject of this thesis but also with the area of automated reasoning in general.

The material from Section 11 is not only art pour l'art of mathematical beauty but it can also open the doors for practical applications. Algebraic structures can be implemented in automated theorem provers, and especially a formalisation in first order logic gives hope for a useful application without to much human interaction. For semirings and the related concept of Kleene algebra (see e.g. [Koz90] or [Koz94]) this was already investigated in work like [Höf09] and [HS07]. [HS08] gives as a more special example a framework for automated reasoning on relations. In [Höf09] it was shown how safety and liveness properties of hybrid systems can be verified using Prover9 (see [McC]), one of the fastest automated theorem provers for first order logic.

Of course, the aim of deployment of automated theorem provers is not to execute an algorithm. The idea is rather to decide whether a certain property is compatible with our approach. Therefore we will develop a framework in first order logic which allows us to reason about bisimulations and related topics.

As automated proof system we decided to use Prover9. This decision is due to the huge amount of preliminary work done for example in [Höf09]. There a lot of theorems

were already proved; we will reprove some of them here because we use a slightly different setting.

12.1 Formalisation in Prover9/Mace4

Prover9 (see [McC]) was developed by W.W. McCune who passed away in 2011. It is an automated prover for first order logic, coupled with Mace4, a searcher for finite models and counterexamples. Its predecessor Otter won a section of the prestigious CADE ATP System Competition (see [CASa] and [CASb]).

The files produced during the experiments with Prover9 and Mace4 can be found at [Glüa] (in English) and [Glüb] (in German). As exact description of the structure is given in Appendix A.

We will illustrate the syntax of Prover9 and Mace4 on the definitions from Section 11 and develop simultaneously a framework for our purposes.

The input files for both Prover9 and Mace4 have the same syntax. Typically, a file starts with the definition of the used operators and is followed by some technical instructions such as the maximal running time. The next two parts list the axioms (assumptions) and the goal to be proved. Line comments can be produced by the sign `%`. We omit the details and concentrate on the essentials.

Figure 12.1 gives the basic part of a formalisation in Prover9 of an idempotent semiring with tests, cf. Definition 11.2.1 and Definition 11.2.2. First, the addition and multiplication operators are introduced, together with their precedence. Every statement (formula, assumption) has to be ended with a full stop. Note that smaller numbers denote higher precedence, so here multiplication binds stronger than addition.

The first block of assumptions states that addition is a commutative and associative binary operation with neutral element 0. Variables starting with `u`, `v`, `w`, `x`, `y` or `z` are by default universally quantified, so the first line of this block corresponds to $\forall$`x,y:x+y=y+x`. In contrast, variables starting with other letters are by default under the reign of an existential quantifier. So after introducing some other laws concerning addition, multiplication and order, in the definition of tests the universal quantifiers are indispensable. Here the function symbol `c` denotes the complement of a test. In order to make it a total function we have to assign also to every non-test a function value which can be chosen arbitrarily.

After the assumptions one can in the goal block state the claims which should be proved. It is possible to formulate more than one claim. Run on it, Prover9 tries to find proofs for the given claims. The proof can be viewed and if one wants also be stored. If Prover9 does not find a proof in reasonable time one can try to run the counterexample searcher Mace4 on the same file. It will search for an interpretation

```
% Language Options
op(500, infix, "+").
op(490, infix, ";").


% commutative additive monoid
x + y = y + x.
x + 0 = x.
x + (y + z) = (x + y) + z.


% multiplicative monoid
x;1 = x & 1;x = x.
x;(y;z) = (x;y);z.


% annihilation laws
0;x = 0 & x;0 = 0.


% idempotency
x + x = x.


% distributivities
x;(y + z) = x;y + x;z.
(x + y);z = x;z + y;z.


% natural order
leq(x,y) <-> x + y = y.


% standard axioms for tests
test(0) & test(1).
all p all q (test(p) & test(q) -> c(p+q) = c(p);c(q) &
      c(p;q) = c(p)+c(q)).
all p (test(p) -> p;c(p) = 0 & p+c(p) = 1).
all p (test(p) -> c(c(p))=p).


% additional axiom, since c() has to be a total function
-test(x) -> c(x) = 0.
```

Figure 12.1: Assumptions Part of a Prover9 Inputfile for Idempotent Semirings with Tests

of the assumptions which makes the claims false. If there is no goal Mace4 will find a model satisfying all assumptions, if possible. However, it will find only finite counterexamples and models.

So if we choose the obviously true goal `(x + x) + y = x + (y + x)`, Prover9 will find a proof immediately. However, if we choose the false goal `x;y = y;x`, Prover9 will not find a proof (it will not terminate) but Mace4 finds a counterexample with four elements in almost instant time. This can be found in the files `x_dot_y_equ_y_dot_x.in`, `x_dot_y_equ_y_dot_x.model`, `x_plus_x_plus_y_equ_x_plus_y_plus_x.in` and `x_plus_x_plus_y_equ_x_plus_y_plus_x.proof` from the folder `misc` (see again Appendix A).

The axioms from Figure 12.1 do not cover all the material we used in Section 11 so we extended it by the axioms from Figure 12.2 which deal with atomic tests and their properties.

The definition of an atomic test is a straightforward translation of Definition 11.2.1 and Definition 11.2.2. The predicate `elem(x,y)` was introduced as an abbreviation for the fact that `x` is an atomic test, `y` is a test and `x` is at most `y`. In the relational interpretation of semirings with tests as presented in Section 11 this predicate corresponds to the element relation which explains the naming.

The next axiom states that the set of tests is atomic, i.e. that every test equals the supremum of all atomic test which are lowerequal. If we use the following characterisation of the supremum

$$x = \sup(A) \Leftrightarrow \forall y : (x \leq y \Leftrightarrow \forall a \in A : a \leq y)$$

we obtain the formula from Figure 12.2. It is interesting that this corresponds to the elementary set theoretic characterisation $A \subseteq B \Leftrightarrow \forall\, x : x \in A \Rightarrow x \in B$.

Using this characterisation, we proved some basic set theoretic theorems which can be found in the folder `atest/`. E.g., the elementary property $x \in \overline{A} \Leftrightarrow x \notin A$ is covered by the file `atest_z_and_test_x_impl_elem_z_c_x_iff_not_elem_z_x.in`.

In this case we used some lemmata, i.e., claims proved in other files. The names of these additional files can be found at the beginning of the assumptions part and the actual theorems together with their names again at the end of this part after the line `% lemmata`. In this special case we split the equivalence `elem_z_c_x_iff_not_elem_z_x` of the theorem we want to prove into two implications. This is a well-known approach in such cases because Prover9 often shows weakness if it is confronted with equivalences. Sometimes this can be avoided by splitting $A \Leftrightarrow B$ into $A \Rightarrow B$ and $B \Rightarrow A$. If both implications can be proved they can be added to the input file and the final proof will be found.

The formalisation of the modal operators is shown in Figure 12.3. We decided to omit

```
% atomic test
atest(x) <-> (-(x=0) & test(x) &
      (all p ((test(p) & leq(p,x) & p!=0) -> p=x))).


% atomicity of test set
(test(x) & test(y)) ->
      (leq(x,y) <-> (all p (atest(p) -> (leq(p,x) -> leq(p,y))))).


% element relation
elem(x,y) <-> (atest(x) & test(y) & leq(x,y)).
```

Figure 12.2: Properties of Atomic Tests in Prover9

the definition of the box operators because they were not of interest in the further course, and an abundance of axioms can lead Prover9 into blind alleys by creating too many superfluous clauses.

The definition of the diamond operators deviates from the one in Definition 11.4.1. We did it in Prover9 via the so called *domain* and *codomain* operators. For a semiring S the domain operator $\ulcorner\cdot : S \to \mathsf{test}(S)$ is a function which assigns to every semiring element a test, defined by the following axioms:

- $\forall x \in S\ \forall p \in \mathsf{test}(S) : \ulcorner(px) = p \cdot \ulcorner x$
- $\forall x \in S : (\ulcorner x) \cdot x = x$
- $\forall x, y \in S : \ulcorner(xy) = \ulcorner(x \cdot \ulcorner y)$

Symmetrically, the codomain operator $\urcorner : S \to \mathsf{test}(S)$ is defined by the following axioms:

- $\forall x \in S\ \forall p \in \mathsf{test}(S) : (xp)\urcorner = (x\urcorner) \cdot p$
- $\forall x \in S : x \cdot x\urcorner = x$
- $\forall x, y \in S : (xy)\urcorner = ((x\urcorner) \cdot y)\urcorner$

In the semiring of relations the domain and codomain model the preimage and the image of a relation, resp. Using these operators the forward and backward diamonds can be defined via the following equations:

```
% axioms for domain
test(dom(x)).
all p (test(p) -> dom(p;x) = p;dom(x)).
dom(x);x = x.
dom(x;y) = dom(x;dom(y)).


% axioms for codomain
test(cod(x)).
all p (test(p) -> cod(x;p) = cod(x);p).
x;cod(x) = x.
cod(x;y) = cod(cod(x);y).


% forward diamond
all p (test(p) -> fd(x,p) = dom(x;p)).
-test(y) -> fd(x,y) = 0.


% backward diamond
all p (test(p) -> bd(x,p) = cod(p;x)).
-test(y) -> bd(x,y) = 0.
```

Figure 12.3: Modal Operators in Prover9

- $|x\rangle p = \ulcorner(xp)$
- $\langle x|p = (px)\urcorner$

The second axioms for the domain and codomain operator in Figure 12.3 are necessary to make them well defined total functions.

These two definitions of the diamond operators (Definition 11.4.1 and the definition from Figure 12.3 are indeed equivalent (see e.g. [DMS06]). In the folder `misc` the reader can find proofs that the given definition implies the one from Definition 11.4.1. We used these properties in some other cases.

Now we will formalise bisimulations in Prover9, as shown in Figure 12.4. Here we deviate from Chapter 11 because the approach presented there contains a lot of material which can not be expressed in first order logic and hence is not suitable as input for Prover9.

In our formalisation we will only consider bisimulations between unlabelled graphs.

```
% carrier function
carr(x) = dom(x) + cod(x).


% z is bisimulation between x and y
bisim(z,x,y) <-> (
% z is left- and righttotal
dom(z) = carr(y) & cod(z) = carr(x) &
% z is simulation between x und y
all p(test(p) -> leq(fd(z,fd(x,p)) , fd(y,fd(z,p)))) &
% converse of z is simulation between y und x
all p(test(p) -> leq(bd(z,fd(y,p)) , fd(x,bd(z,p))))).


% bisimilarity of x and y
bisimilar(x,y) <-> exists a (bisim(a,x,y)).
```

Figure 12.4: Carrier Function and Bisimulation in Prover9

Then a bisimulation between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a left and right total relation $B \subseteq V_1 \times V_2$ with the following properties:

- $B^\circ; E_1 \subseteq E_2; B^\circ$
- $B; E_2 \subseteq E_1; B$

These requirements are the same as in Definition 4.2.1 and the subsequent discussion. In our formalisation we found it a little bit more convenient to consider a relation between V_2 and V_1 (the variable z in Figure 12.4) which does not change a lot since we reason simply about B° instead of B.

Left and right totality are expressed via the carrier function which is simply the sum of the domain and codomain (see the first and second line of Figure 12.4). This does not express exactly totality in the relational sense: if G_1 or G_2 contains isolated nodes they will be 'overlooked' by the carrier function. So our definition in Figure 12.4 is suitable only for systems without isolated nodes. This is no great restriction because in most cases we analysed isolated nodes were not of interest.

However, if we want to deal with the expansion operation we have to take care of this fact. After refining a model it can happen that the refined model contains isolated nodes (cf. the refinement we computed in Subsection 7.3.1: there a and b were isolated). So if we want to formalise the expansion operation analogously to Theorem 6.3.2 we have to adapt the term of bisimulation (this takes place in the definition of a comprehensive bisimulation). This leads to the definitions given in Figure 12.5.

```
% z is comprehensive bisimulation between x and y
comp_bisim(z,x,y) <-> (
% z comprehends both carriers
leq(carr(y),dom(z)) & leq(carr(x),cod(z)) &
% z is simulation between x und y
all p(test(p) -> leq(fd(z,fd(x,p)) , fd(y,fd(z,p)))) &
% converse of z is simulation between y und x
all p(test(p) -> leq(bd(z,fd(y,p)) , fd(x,bd(z,p))))).


% x1 is expansion of y1
exp(x1,y1,x,y,z) <-> (
% z is bisimulation between x and y,
% x1 and y1 are submodels of x and y, resp., and
% z is comprehensive bisimulation between x1 and y1
bisim(z,x,y) & leq(x1,x) & leq(y1,y) & comp_bisim(z,x1,y1) &
% x1 is the greatest model with these properties
all x2(leq(x2,x) & comp_bisim(z,x2,y1) -> leq(x2,x1))).
```

Figure 12.5: Expansion in Prover9

12.2 Experiments with Prover9

We used these formalisations for some experiments with Prover9. As mentioned above, the results can be found under [Glüa] (in English) and [Glüb] (in German). An extensive documentation of the results is the content of Appendix A.

12.2.1 Approach and Additional Predicates

Working with Prover9 holds some surprises, both pleasant and less pleasant ones. In some cases Prover9 discovers difficult proofs in amazingly short time, in other cases it does not succeed in proving simple facts. Often it shows problems if the claim which is to be proven has the form $A \Rightarrow B \wedge C$. Even in simple cases it can happen that it searches a proof for hours without finding it. This behaviour can be cured in a lot of cases if one proves both $A \Rightarrow B$ and $A \Rightarrow C$ separately, then adds these two claims to the set of support and starts a new run. There are cases where Prover9 does not find a proof even under these assumptions because it concentrates to much on other axioms. The radical remedy is to remove all others formulas except $A \Rightarrow B$ and $A \Rightarrow C$. A good example for this approach can be found in the file `atest_z_and_test_x_impl_elem_z_c_x_iff_not_elem_z_x.in` from the

folder `atest`.
The theorems we proved are listed in Appendix A. We briefly describe some additional predicates we introduced.

- The predicate `bisimilar(x,y)` is defined by

  ```
  bisimilar(x,y) <-> exists a (bisim(a,x,y)).
  ```

  It simply states that `x` and `y` are bisimilar.

- The predicate `live(x)` is defined by

  ```
  live(x) <-> leq(cod(x),dom(x)).
  ```

  In the relational setting, it means that every node with an ingoing edge has also an outgoing edge.

- The predicate `lp(x1,x)` is defined by

  ```
  lp(x1,x) <-> leq(x1,x) & live(x1).
  ```

  This means that `x1` is live and a submodel of `x`, i.e., `x1` is a live part of `x`.

- The predicate `glp(x1,x)` is defined by

  ```
  glp(x1,x) <-> (lp(x1,x) & all x2(lp(x2,x) -> leq(x2,x1))).
  ```

  It states that `x1` is the greatest live part of `x`. In the terminology of models, this means that `x1` is the greatest live submodel of `x`.

- The predicate `pscon(x,y)` is defined by

  ```
  pscon(x,y) <-> (all p(test(p) -> fd(x,p) = bd(y,p))).
  ```

  It corresponds to Definition 11.7.2.

12.2.2 Experimental Results

As one can see in Appendix A, we could prove a lot of properties using Prover9. However, this was possible only with human interaction. This is a well-known problem, e.g. in [Höf09] most of the proofs demanded human intervention by addition or removal of theorems or splitting equalities into two inequalities. The general problem seems to be that Prover9 does not have a kind of human intuition at its disposal and can not distinguish between useful and less useful directions of search.

Problems of this kind were and are discussed at numerous occasions (see [IJC] or [CAD]) so we will not go into detail. We summarise that Prover9 is not well suitable for automated reasoning in our area but it is a valuable tool for interactive verification. Other conclusions will be discussed in the concluding Section 13.2.

Chapter 13

Conclusion

Finally, we take a look back and summarise the material, and take a look ahead onto future work and sketch further ideas for subsequent research. Some questions remained open or arose during the work on this thesis which will also be discussed.

13.1 Summary

The main purpose of this thesis was to investigate an approach for model refinement using bisimulation quotients. Starting from a simple and ad hoc solved puzzle in Chapter 3 we developed a comprehensive framework for a widespread variety of algorithmic problems. It covers both qualitative problems like in Chapter 7 as well as quantitative aspects like in Chapters 8 and 10 and theoretical considerations like in Chapter 9. The problem area from Chapter 7 is closely related to temporal logic and has applications in control theory and the theory of hybrid systems. In Chapter 8 we developed a general framework for cost functions in a set-labelled graph which can serve to describe a lot of optimality problems in graphs such as shortest walks, maximum reliability and related terms. Stochastic games, as investigated in Chapter 10, arise in the context of fuzzy decision making.

Our basic idea was to solve a problem not on an original instance but on an instance with equivalent dynamic behaviour but possibly smaller size. This was achieved by

means of bisimulation equivalences, a relation on the nodes of a transition system. Under certain circumstances, we also could tackle infinite systems if the constructed equivalent system has a finite number of states. Because semirings are a well suitable tool for algebraic reasoning about relations we could also give an algebraic approach via the theory of semirings in Chapter 11 and used the tools we developed there to employ automated theorem provers as described in Chapter 12.

For every refinement algorithm under consideration we investigated its compatibility with bisimulation equivalences and analysed the complexity of the quotient-based variant as sketched generically in Algorithm 1. In most cases it was possible to show a potential speed-up.

Besides the investigation of that approach we dealt with related problems. In Subsection 4.1.2 we proposed a data structure for set-labelled graphs which we used in later chapters in the implementation of refinement algorithms and the analysis of their running time. General problems of refineability were considered in Sections 8.4 and 8.5. There we investigated under which circumstances target models are refinable.

13.2 Future Work and Open Questions

For every algorithm we analysed its complexity and analysed whether it can lead to a speed-up compared to the immediate application of an algorithm to the original system. In most cases, we could show the possibility of a speed-up. However, we did not test our approach on real world instances, so the next step should be the test on examples from practice.

Another branch of future work should search for a more general characterisation of problems which can be treated with our approach. For example, we did not tackle $\mathcal{NP}$-complete problems. On the other hand, problems in $\mathcal{P}$ (Chapters 7 and 8) or even in $\mathcal{NP} \cap \mathsf{co}\mathcal{NP}$ (Chapter 10) are compatible with bisimulation equivalences. For some classes of qualitative problems as CTL^* or acceptance of finite automata, there are already results concerning bisimulations. However, a general characterisation scheme seems to be unknown and will be a challenge for future research in logic. A similar branch of research should be the application of interactive higher-order theorem provers like Isabelle or Coq instead of Mace4 as in Chapter 12. This will also open the door for more practical applications.

In the context of Chapter 7 we dealt with a set F of 'good' nodes and investigated properties involving this set. There are other related problems which besides F consider a second set G and formalise properties of the form $\Box F \to \Diamond G$ and deal with similar formulae from temporal logic. An approach analogous to the one from Chapter 7 should work in this cases, too, but needs to be analysed.

In Chapter 8 we showed in Theorem 8.2.2 the existence of optimal walks in a target model with finite label set and an associated cumulative s-dioid. However, this does not mean immediately that a target model with these properties is refineable. The intuition is that it is refineable but an exact proof is still missing.

The original formulation of Stochastic Games as given in [Sha53] is more general than the notion of SSGs we used in Chapter 10 (although the two terms are equivalent in a certain sense). The question arises whether an adapted version of the algorithms from Chapter 10 are also applicable to Stochastic Games as defined in [Sha53]. Intuitively, this should be possible but it will require exact and thorough elaboration.

The formalisation from Chapter 11 used a semiring with atomic test set, and some of the proofs used this property. In Chapter 12 we used atomic tests only for theorems concerning set theory. The question arises whether some theorems from Chapter 11 hold also if the atomicity of the test set is dropped (provided a reasonable definition of the properties under consideration without atomic tests is possible). Mace4 is no suitable tool for finding a counterexample because it can only find finite models. But clearly, in every finite semiring S, $\mathsf{test}(S)$ is atomic.

An extension of modal semirings is the concept of a Kleene algebra with domain (see [DMS06]) which captures finite iteration by the Kleene star operation. It is well known that the Kleene star can be used to model optimality problems as in Chapter 8. So it seems to be promising to integrate the Kleene star operation into the framework developed in Chapter 11. The same holds for the Omega operator (see [Coh00]) which models infinite iteration. It should also be tried to extend the framework from Chapter 11 in such a way that it can also describe labelled transition systems (recall that the running example from Page 99 was a simple relation without edge labels). This can lead to an algebra suitable for verification of practical applications derived from the framework we developed in Chapter 8.

Appendix A

Results of Prover9

This chapter is dedicated to the experiments we did with Prover9. We explain the structure of the files and summarise the results in table form.

A.1 Namings

In general, input files are characterised by the extension `.in`, proofs generated by Prover9 by `.proof` and counterexamples found by Mace4 by `.model`. Associated files have the same file name but different extensions, so `<filename>.proof` is the proof generated by Prover9 run on the input file `<filename>.in`. Analogously, if Mace4 finds a counterexample for the input file `<filename>.in` it can be found in `<filename>.model`. The actual file name is derived from the goal by omitting all quantifiers and parentheses. Every implication is substituted by `impl`, every reverse implication ($\Leftarrow$) by `isimpl`, every equivalence by `iff`, every addition by `plus` and every multiplication by `dot`. Conjunction and disjunction become `and` and `or`, resp. Equality is denoted by `equ`, less- or equality by `leq` and inequality by `neq`. The constants 0 and 1 are represented by `zero` and `one`, resp. For better readability, the name is structured by underlines. Predicate names like `bisim` or others (see later) remain untouched. So an input file with the goal `bisimilar(x,y) & live(x) -> live(y)` is named `bisimilar_x_y_and_live_x_impl_live_y.in` (see the folder `live`).

There may be exceptions from these rules if the claim can be characterised otherwise in a concise way. So the input file with the goal `cod(x+y) = cod(x) + cod(y)` is named `codomain_is_additive`.

A.2 Folder Structure

The folders are organised as follows:

- `atest`: This folder deals with properties of atomic tests (which correspond to set theoretic theorems). In the associated Table A.1 we use the element sign $\in$ both for the traditional element relation (as in $x, y \in \mathsf{atest}$) and for the `elem` predicate we introduced in Chapter 12 (as in $z \in \overline{x}$).
- `expansion`: This folder deals with the expansion operation as defined in Figure 12.5. Among other properties, we showed that the expansion of a live model is also live.
- `id_bisim`: This folder shows that the identity on the carrier set (modelled by `carr(x)`) is a bisimulation.
- `live`: This folder is about the `live`-predicate and its derivatives `lp` and `glp` (see Subsection 12.2.1).
- `misc`: This folder contains lemmata we use in a lot of input files. It involves properties of the natural order, test properties, properties of the modal operators and an alternative characterisation of the diamond and box operators (cf. Definition 11.4.1). In contrast to the other folders the imported lemmata are completely listed. A special case is the file `x_dot_y_equ_y_dot_x.in` because there is no corresponding `.proof`-file but a `.model`-file with the same name (cf. Section 12.1) and is not listed in the corresponding table.
- `pscon`: This folder's main theorem shows that `pscon(x,y)` implies `pscon(y,x)`. It is used in the folder `symm_bisim`.
- `sum_bisim`: This folder's main theorem shows that bisimulations are closed under finite sum (`bisim(z1,x,y)` $\wedge$ `bisim(z2,x,y)` $\Rightarrow$ `bisim(z1+z2,x,y)`). In a quantale, bisimulations are closed under arbitrary sum but due to the limitations of first order logic this could not be used as input for Prover9.
- `symm_bisim`: This folder contains the proof that the bisimilarity relation is symmetric. More precisely, it shows that `bisim(x,z1,z2)` and `pscon(x,y)` imply `bisim(y,z2,z1)`.

- `top`: This folder deals with some properties of the greatest element $\top$.
- `trans_bisim`: This folder shows that the bisimulation relation is transitive. The main theorem of this folder is `bisim(x12,z1,z2)` $\wedge$ `bisim(x23,z2,z3)` $\Rightarrow$ `bisim(x23;x12,z1,z3)`.

A.3 Table Structure

Each table consists of four columns with the following namings and meanings:

- name: This column simply gives the name of the file in the corresponding folder without its extension. As explained in Section A.1, `<name>.in` is an input file for Prover9, `<name>.proof` is the corresponding proof given by Prover9 and `<name>.model` contains a counterexample produced by Mace4.
- theorem: This column contains the claim of the corresponding file in a better human-readable writing. By default, are variables are universally quantified. The notations are the same as in Chapters 11 and 12. Occasional deviations are explained in Section A.2.
- additional theorems: This column's entry equals 'no' if no additional theorems beside the basic definitions given in Chapter 12 were used. The entry equals 'misc' if only theorems from the `misc`-folder were inserted. An entry 'yes' indicates the use of additional theorems (see also Chapter 12 about the imports). We did not indicate the cases where axioms were removed.
- time: This column gives the time in seconds Prover9/Mace4 took for the proof/-counterexample. The test runs were done under Linux using an Intel(R) Pentium(R) 4 CPU 2.80GHz and version 2009-02A of Prover9 from February 2009 (the input files were created using the GUI from [McC] which comes along with the version 0.5 of Prover9Mace4 from December 2007). Possibly, other environments can lead to slightly different results. The values are rounded to the next integer.

A.4 Tabular Results

The following pages show the content of the folders as explained in the previous Section A.3.

Table A.1: Content of `atest`

name	theorem	additional theorems	time
atest_x_and_atest_y_and_x_neq_y_impl_ not_atest_x_plus_y	$x, y \in \mathsf{atest} \wedge x \neq y \Rightarrow x + y \notin \mathsf{atest}$	no	0 s
atest_x_and_atest_y_impl_ x_dot_y_equ_x_or_x_dot_y_equ_zero	$x, y \in \mathsf{atest} \Rightarrow x \cdot y = x \vee x \cdot y = 0$	no	3 s
atest_z_and_test_x_impl_ elem_z_c_x_iff_not_elem_z_x	$z, x \in \mathsf{atest} \Rightarrow (z \in \neg x \Leftrightarrow z \notin x)$	yes	0 s
atest_z_and_test_x_impl_ elem_z_c_x_impl_not_elem_z_x	$z, x \in \mathsf{atest} \Rightarrow (z \in \neg x \Rightarrow z \notin x)$	yes	1 s
atest_z_and_test_x_impl_ elem_z_x_iff_z_dot_x_equ_z	$z, x \in \mathsf{atest} \Rightarrow (z \in x \Leftrightarrow z \cdot x = z)$	yes	1 s
atest_z_and_test_x_impl_ elem_z_x_impl_not_elem_z_c_x	$z \in \mathsf{atest} \wedge x \in \mathsf{test} \Rightarrow (z \in x \Rightarrow z \notin \neg x)$	yes	2 s
atest_z_and_test_x_impl_ not_elem_z_x_impl_elem_z_c_x	$z \in \mathsf{atest} \wedge x \in \mathsf{test} \Rightarrow (z \notin x \Rightarrow z \in \neg x)$	yes	39 s
atest_z_and_test_x_impl_ not_elem_z_x_impl_z_dot_x_equ_zero	$z \in \mathsf{atest} \wedge x \in \mathsf{test} \Rightarrow (z \notin x \Rightarrow z \cdot x = 0)$	yes	4 s
different_sets_contain_different_elements	$x, y \in \mathsf{test} \wedge x \neq y \Rightarrow \exists a : a \in x \wedge a \notin y \vee a \in y \wedge a \notin x$	no	4 s
disjoint_sets_contain_different_elements	$x, y \in \mathsf{test} \wedge x \cdot y = 0 \Rightarrow (z \in x \Rightarrow z \notin y)$	no	6 s
test_x_and_test_y_impl_ leq_x_y_iff_x_dot_y_equ_x	$x, y \in \mathsf{test} \Rightarrow (x \leq y \Leftrightarrow x \cdot y = x)$	yes	10 s
test_x_impl_test_c_x	$x \in \mathsf{test} \Rightarrow \neg x \in \mathsf{test}$	no	0 s

name	theorem	additional theorems	time
bisim_z_x_y_impl_comp_bisim_z_x_y	$bisim(z,x,y) \Rightarrow comp\_bisim(z,x,y)$	yes	0 s
bisim_z_x_y_impl_exp_x_y_x_y_z	$bisim(z,x,y) \Rightarrow exp(x,y,x,y,z)$	yes	0 s
bisim_z_x_y_impl_ leq_carr_x_cod_z	$bisim(z,x,y) \Rightarrow carr(x) \leq \overline{z}$	misc	0 s
bisim_z_x_y_impl_ leq_carr_y_dom_z	$bisim(z,x,y) \Rightarrow carr(y) \leq \ulcorner z$	misc	0 s
comp_bisim_z_x_y_and_live_y_impl_ live_x	$comp\_bisim(z,x,y) \wedge live(y) \Rightarrow live(x)$	yes	336 s
expansion_is_unique	$exp(x1,y1,x,y,z) \wedge exp(x2,y1,x,y,z) \Rightarrow x1 = x2$	no	3 s
expansion_of_live_is_live	$exp(x1,y1,x,y,z) \wedge live(y1) \Rightarrow live(x1)$	yes	0 s
test_x_and_leq_carr_y_x_impl_ x_dot_y_equ_y_and_y_dot_x_equ_y	$test(x) \wedge carr(y) \leq x \Rightarrow x \cdot y = y \wedge y \cdot x = y$	yes	3 s
test_x_and_leq_cod_y_x_impl_ y_dot_x_equ_y	$test(x) \wedge \overline{y} \leq x \Rightarrow y \cdot x = y$	misc	2 s
test_x_and_leq_dom_y_x_impl_ x_dot_y_equ_y	$test(x) \wedge \ulcorner y \leq x \Rightarrow x \cdot y = y$	misc	1 s

Table A.2: Content of `expansion`

Table A.3: Content of `id_bisim`

name	theorem	additional theorems	time
bisim_carr_x_x_x	$bisim(carr(x), x, x)$	yes	0 s
cod_carr_x_equ_carr_x	$(carr(x))^{\urcorner} = carr(x)$	misc	1 s
dom_carr_x_equ_carr_x	$^{\ulcorner}(carr(x)) = carr(x)$	misc	1 s
leq_fd_carr_x_bd_x_p_ fd_x_bd_carr_x_p	$\|carr(x)\rangle\langle x\|p \leq \|x\rangle\langle carr(x)\|p$	misc	12 s
leq_fd_carr_x_fd_x_p_ fd_x_fd_carr_x_p	$\|carr(x)\rangle\|x\rangle p \leq \|x\rangle\|carr(x)\rangle p$	yes	3 s

name	theorem	additional theorems	time
bisimilar_x_y_and_live_x_impl_live_y	$bisimilar(x, y) \wedge live(x) \Rightarrow live(y)$	misc	17 s
bisimilar_x_y_and_live_y_impl_live_x	$bisimilar(x, y) \wedge live(y) \Rightarrow live(x)$	yes	0 s
bisim_z_x_y_and_live_y_impl_live_x	$bisim(z, x, y) \wedge live(y) \Rightarrow live(x)$	misc	2 s
glp_is_unique	$glp(x1, x) \wedge glp(x2, x) \Rightarrow x1 = x2$	misc	0 s
live_x_and_live_y_impl_live_x_plus_y	$live(x) \wedge live(y) \Rightarrow live(x + y)$	misc	0 s
live_x_impl_glp_x_x	$live(x) \Rightarrow glp(x, x)$	misc	0 s
lp_x1_x_and_lp_x2_x_impl_lp_x1_plus_x2_x	$lp(x1, x) \wedge lp(x2, x) \Rightarrow lp(x1 + x2, x)$	misc	0 s

Table A.4: Content of `live`

Table A.5: Content of `misc` (first part)

name	theorem	additional theorems	time
additivity_bd_1	$\langle x+y\vert p = \langle x\vert p + \langle y\vert p$	yes	0 s
additivity_bd_2	$\langle x\vert (p+q) = \langle x\vert p + \langle y\vert p$	yes	1 s
additivity_fd_1	$\vert x+y\rangle p = \vert x\rangle p + \vert y\rangle p$	yes	0 s
additivity_fd_2	$\vert x\rangle (p+q) = \vert x\rangle p + \vert y\rangle p$	yes	1 s
bd_y_dot_x_p_equ_ bd_x_bd_y_p	$\langle y\cdot x\vert p = \langle x\vert\langle y\vert p$	no	0 s
codomain_is_additive	$(x+y)^\urcorner = x^\urcorner + y^\urcorner$	yes	2 s
codomain_is_ least_right_neutral	$p \in \mathsf{test} :$ $(x\cdot p = x \Leftrightarrow x^\urcorner \leq p)$	yes	57 s
cod_p_equ_p	$p \in \mathsf{test} : p^\urcorner = p$	yes	1 s
complement_is_antitone	$p,q \in \mathsf{test} : p \leq q \Rightarrow \neg q \leq \neg p$	no	6 s
complement_is_antitone_iff	$p,q \in \mathsf{test} : p \leq q \Leftrightarrow \neg q \leq \neg p$	yes	0 s
complement_is_test	$p \in \mathsf{test} \Rightarrow \neg p \in \mathsf{test}$	no	0 s
domain_is_additive	$^\ulcorner(x+y) = {}^\ulcorner x + {}^\ulcorner y$	yes	2 s
domain_is_ least_left_neutral	$p \in \mathsf{test} :$ $(p\cdot x = x \Leftrightarrow {}^\ulcorner x \leq p)$	yes	57 s
dom_p_equ_p	$p \in \mathsf{test} : {}^\ulcorner p = p$	yes	1 s
fd_x_dot_y_p_equ_ fd_x_bd_y_p	$\vert x\cdot y\rangle p = \vert x\rangle\vert y\rangle p$	no	0 s

name	theorem	additional theorems	time
greatest_left_annihilator	$\neg p \cdot x = 0 \Leftrightarrow \lceil x \leq p$	yes	68 s
greatest_right_annihilator	$x \cdot \neg p = 0 \Leftrightarrow \vec{x^\rceil} \leq p$	yes	56 s
isotonies	$x \leq y \Rightarrow xz \leq yz \wedge zx \leq zy \wedge x + z \leq y + z$	no	1 s
leq_bd_x_p_c_q_iff_ p_dot_x_dot_q_equ_zero	$\langle x \vert p \leq \neg q \Leftrightarrow p \cdot x \cdot q = 0$	yes	33 s
leq_bd_x_p_c_q_impl_ p_dot_x_dot_q_equ_zero	$\langle x \vert p \leq \neg q \Rightarrow p \cdot x \cdot q = 0$	yes	1 s
leq_bd_x_p_c_q_isimpl_ p_dot_x_dot_q_equ_zero	$\langle x \vert p \leq \neg q \Leftarrow p \cdot x \cdot q = 0$	yes	0 s
leq_carr_x_p_impl_ p_dot_x_equ_x_and_ x_dot_p_equ_x	$p \in \mathsf{test} : carr(x) \leq p \Rightarrow px = x \wedge xp = x$	no	16 s
leq_cod_x_p_impl_ x_dot_p_equ_x	$p \in \mathsf{test} : \vec{x^\rceil} \leq p \Rightarrow xp = x$	no	16 s
leq_cod_x_plus_y_p_iff_ leq_cod_x_plus_cod_y_p	$p \in \mathsf{test} : \vec{(x+y)^\rceil} \leq p \Leftrightarrow \vec{x^\rceil} + \vec{y^\rceil} \leq p$	yes	0 s
leq_cod_x_plus_y_p_impl_ leq_cod_x_plus_cod_y_p	$p \in \mathsf{test} : \vec{(x+y)^\rceil} \leq p \Rightarrow \vec{x^\rceil} + \vec{y^\rceil} \leq p$	yes	4 s
leq_cod_x_plus_y_p_isimpl_ leq_cod_x_plus_cod_y_p	$p \in \mathsf{test} : \vec{(x+y)^\rceil} \leq p \Leftarrow \vec{x^\rceil} + \vec{y^\rceil} \leq p$	yes	0 s

Table A.6: Content of `misc` (second part)

Table A.7: Content of `misc` (third part)

name	theorem	additional theorems	time
leq_dom_x_p_impl_ p_dot_x_equ_x	$p \in \mathsf{test} : \ulcorner x \leq p \Rightarrow px = x$	no	8 s
leq_dom_x_plus_y_p_iff_ leq_dom_x_plus_dom_y_p	$p \in \mathsf{test} : \ulcorner(x+y) \leq p \Leftrightarrow \ulcorner x + \ulcorner y \leq p$	yes	0 s
leq_dom_x_plus_y_p_impl_ leq_dom_x_plus_dom_y_p	$p \in \mathsf{test} : \ulcorner(x+y) \leq p \Rightarrow \ulcorner x + \ulcorner y \leq p$	yes	4 s
leq_dom_x_plus_y_p_isimpl_ leq_dom_x_plus_dom_y_p	$p \in \mathsf{test} : \ulcorner(x+y) \leq p \Leftarrow \ulcorner x + \ulcorner y \leq p$	yes	0 s
leq_fd_x_q_c_p_iff_ p_dot_x_dot_q_equ_zero	$\|x\rangle q \leq \neg p \Leftrightarrow p \cdot x \cdot q = 0$	yes	30 s
leq_fd_x_q_c_p_impl_ p_dot_x_dot_q_equ_zero	$\|x\rangle q \leq \neg p \Rightarrow p \cdot x \cdot q = 0$	yes	1 s
leq_fd_x_q_c_p_isimpl_ p_dot_x_dot_q_equ_zero	$\|x\rangle q \leq \neg p \Leftarrow p \cdot x \cdot q = 0$	yes	0 s
leq_is_antisymmetric	$px = y \Leftrightarrow x \leq y \wedge y \leq x$	no	5 s
leq_is_transitive	$x \leq y \wedge y \leq z \Rightarrow x \leq z$	no	0 s
leq_p_dot_q_q_dot_p leq_cod_x_plus_cod_y_p	$p, q \in \mathsf{test} : pq \leq qp$	yes	31 s
leq_p_q_iff_p_dot_c_q_equ_zero	$p, q \in \mathsf{test} : p \leq q \Leftrightarrow p \cdot \neg q = 0$	yes	0 s
leq_p_q_impl_p_dot_c_q_equ_zero	$p, q \in \mathsf{test} : p \leq q \Rightarrow p \cdot \neg q = 0$	no	2 s
leq_p_q_isimpl_p_dot_c_q_equ_zero	$p, q \in \mathsf{test} : p \leq q \Leftarrow p \cdot \neg q = 0$	yes	7 s

name	theorem	additional theorems	time
multiplication_on_tests	$p, q \in \mathsf{test} : p \cdot p = p \wedge p \cdot q = q \cdot p$	yes	0 s
sum_is_supremum	$x + y \leq z \Leftrightarrow x \leq z \wedge y \leq z$	no	1 s
test_cod_x	$\overline{x} \in \mathsf{test}$	no	0 s
test_cod_x_plus_cod_y	$\overline{x} + \overline{y} \in \mathsf{test}$	yes	0 s
test_dom_x	$\overline{\lceil x} \in \mathsf{test}$	no	0 s
test_dom_x_plus_dom_y	$\overline{\lceil x} + \overline{\lceil y} \in \mathsf{test}$	yes	0 s
test_fd_x_p_and_test_bd_x_p	$\lvert x\rangle p \in \mathsf{test} \wedge \langle x\rvert p \in \mathsf{test}$	yes	0 s
test_indirect_equality_1	$p, q \in \mathsf{test} :$ $p = q \Leftrightarrow \forall r \in \mathsf{test} : p \leq r \Leftrightarrow q \leq r$	no	0 s
test_indirect_equality_2	$p, q \in \mathsf{test} :$ $p = q \Leftrightarrow \forall r \in \mathsf{test} : r \leq p \Leftrightarrow r \leq q$	no	0 s
test_is_closed	$p, q \in \mathsf{test} : pq \in \mathsf{test} \wedge p + q \in \mathsf{test}$	yes	0 s
test_multiplication_is_commmutative	$p, q \in \mathsf{test} : p \leq q \Leftrightarrow p \cdot \neg q = 0$	yes	1 s
test_multiplication_is_idempotent	$p \in \mathsf{test} : p \cdot p = p$	no	2 s
x_plus_x_plus_y_equ_x_plus_y_plus_x	$(x + x) + y = x + (y + x)$	no	0 s
zero_is_indivisible	$x + y = 0 \Leftrightarrow x = 0 \wedge y = 0$	no	0 s

Table A.8: Content of `misc` (fourth part)

Table A.9: Content of `pscon` (first part)

name	theorem	additional theorems	time
pscon_x_y_impl_bd_x_p_leq_fd_y_p	$pscon(x,y) \Rightarrow \langle x\|p \leq \|y\rangle p$	yes	3 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_1	$\langle x\|p \leq \|y\rangle p \Leftrightarrow p \cdot x \cdot \neg\|y\rangle p = 0$	yes	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_1a	$\langle x\|p \leq \|y\rangle p \Leftarrow p \cdot x \cdot \neg\|y\rangle p = 0$	misc	17 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_1b	$\langle x\|p \leq \|y\rangle p \Rightarrow p \cdot x \cdot \neg\|y\rangle p = 0$	misc	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_2	$\|x\rangle\neg\|y\rangle p \leq \neg p \Leftrightarrow p \cdot x \cdot \neg\|y\rangle p = 0$	misc	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_3	$(\forall q \in \mathsf{test} : \|x\rangle q \leq \langle y\|q) \Rightarrow$ $(\|x\rangle\neg\langle y\|p \leq \neg p \Leftarrow \langle y\|\neg\|y\rangle p \leq \neg p)$	misc	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_4	$\langle y\|\neg\|y\rangle p \leq \neg p \Leftrightarrow \neg\|y\rangle p \cdot y \cdot p = 0$	yes	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_4a	$\langle y\|\neg\|y\rangle p \leq \neg p \Rightarrow \neg\|y\rangle p \cdot y \cdot p = 0$	misc	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_4b	$\langle y\|\neg\|y\rangle p \leq \neg p \Leftarrow \neg\|y\rangle p \cdot y \cdot p = 0$	misc	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_5	$\|y\rangle p \leq \|y\rangle p \Leftrightarrow \neg yp \cdot y \cdot p = 0$	misc	0 s
pscon_x_y_impl_ bd_x_p_leq_fd_y_p_step_6	$\neg\|y\rangle p \cdot y \cdot p = 0$	yes	0 s

name	theorem	additional theorems	time
pscon_x_y_impl_fd_y_p_leq_bd_x_p	$pscon(x,y) \Rightarrow \|y\rangle p \leq \langle x\|p$	yes	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_1	$\|x\rangle p \leq \langle y\|p \Leftrightarrow \neg\langle y\|p \cdot x \cdot p = 0$	yes	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_1a	$\|x\rangle p \leq \langle y\|p \Leftarrow \neg\|y\rangle p \cdot x \cdot p = 0$	misc	15 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_1b	$\|x\rangle p \leq \langle y\|p \Rightarrow \neg\|y\rangle p \cdot x \cdot p = 0$	misc	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_2	$\langle x\|\neg\langle y\|p \leq \neg p \Leftrightarrow \neg\langle y\|p \cdot x \cdot p = 0$	misc	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_3	$(\forall q \in \mathsf{test} : \langle x\|q \leq \|y\rangle q) \Rightarrow$ $(\langle x\|\neg\|y\rangle p \leq \neg p \Leftarrow \|y\rangle\neg\langle y\|p \leq \neg p)$	misc	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_4	$\|y\rangle\neg\langle y\|p \leq \neg p \Leftrightarrow p \cdot y \cdot \neg\|y\rangle p = 0$	yes	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_4a	$\|y\rangle\neg\langle y\|p \leq \neg p \Rightarrow p \cdot y \cdot \neg\|y\rangle p = 0$	misc	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_4b	$\|y\rangle\neg\langle y\|p \leq \neg p \Leftarrow p \cdot y \cdot \neg\|y\rangle p = 0$	misc	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_step_5	$\langle y\|p \leq \langle y\|p \Leftrightarrow p \cdot y \cdot \neg yp = 0$	misc	0 s
pscon_x_y_impl_ fd_y_p_leq_bd_x_p_6	$p \cdot y \cdot \neg\langle y\|p = 0$	yes	0 s
pscon_x_y_impl_pscon_y_x	yes	0 s	

Table A.10: Content of `pscon` (second part)

Table A.11: Content of `sum_bisim` (first part)

name	theorem	additional theorems	time
bd_z1_plus_z2_fd_y_p_equ_ bd_z1_fd_y_p_plus_bd_z2_fd_y_p	$\langle z_1 + z_2 \| \| y \rangle p = \langle z_1 \| \| y \rangle p + \langle z_2 \| \| y \rangle p$	misc	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ bd_z1_plus_z2_fd_y_p_ equ_bd_z1_fd_y_p_plus_bd_z2_fd_y_p	$bisim(z1, x, y) \wedge bisim(z2, x, y) \Rightarrow$ $\langle z1 + z2 \| \| y \rangle p = \langle z1 \| \| y \rangle p + \langle z2 \| \| y \rangle p$	yes	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ bisim_z1_plus_z2_x_y	$bisim(z1, x, y) \wedge bisim(z2, x, y) \Rightarrow$ $bisim(z1 + z2, x, y)$	yes	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ cod_z1_plus_z2_equ_carr_x	$bisim(z1, x, y) \wedge bisim(z2, x, y) \Rightarrow$ $(z1 + z2)^{\urcorner} = carr(x)$	misc	0 s
bisim_z1_x_y_and_bisim_z2_x_y_ dom_z1_plus_z2_equ_carr_y	$bisim(z1, x, y) \wedge bisim(z2, x, y) \Rightarrow$ $^{\ulcorner}(z1 + z2) = carr(y)$	misc	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ fd_x_bd_z1_plus_z2_p_equ_ fd_x_bd_z1_p_plus_fd_x_bd_z2_p	$bisim(z1, x, y) \wedge bisim(z2, x, y) \Rightarrow$ $\|x\rangle\langle z1 + z2\|p = \|x\rangle\langle z1\|p + \|x\rangle\langle z2\|p$	yes	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ fd_y_fd_z1_p_plus_fd_y_fd_z2_p_equ_ fd_y_fd_z1_plus_z2_p	$bisim(z1, x, y) \wedge bisim(z2, x, y) \Rightarrow$ $\|y\rangle\|z1\rangle p + \|y\rangle\|z2\rangle p = \|y\rangle\|z1 + z2\rangle p$	misc	0 s

name	theorem	additional theorems	time
bisim_z1_x_y_and_bisim_z2_x_y_impl_ fd_z1_plus_z2_fd_x_p_equ_ fd_z1_fd_x_p_plus_fd_z2_x_p	$bisim(z1,x,y) \wedge bisim(z2,x,y) \Rightarrow$ $\|z1+z2)\rangle\|x\rangle p = \|z1\rangle\|x\rangle p + \|z2\rangle\|x\rangle p$	misc	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ leq_bd_z1_fd_y_p_plus_bd_z2_fd_y_p_ fd_x_bd_z1_p_plus_fd_x_bd_z2_p	$bisim(z1,x,y) \wedge bisim(z2,x,y) \Rightarrow$ $\langle z1\|\|y\rangle p + \langle z2\|\|y\rangle p \leq$ $\|x\rangle\langle z1\|p + \|x\rangle\langle z2\|p$	yes	0 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ leq_bd_z1_plus_z2_fd_y_p_ fd_x_bd_z1_plus_z2_p	$bisim(z1,x,y) \wedge bisim(z2,x,y) \Rightarrow$ $\langle z1+z2\|\|y\rangle p \leq \|x\rangle\langle z1+z2\|p$	yes	2 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ leq_fd_z1_fd_x_p_plus_fd_z2_fd_x_p_ fd_y_fd_z1_p_plus_fd_y_fd_z2_p	$bisim(z1,x,y) \wedge bisim(z2,x,y) \Rightarrow$ $\|z1\rangle\|x\rangle p + \|z2\rangle\|x\rangle p \leq$ $\|y\rangle\|z1\rangle p + \|y\rangle\|z2\rangle p$	misc	5 s
bisim_z1_x_y_and_bisim_z2_x_y_impl_ leq_fd_z1_plus_z2_fd_x_p_ fd_y_fd_z1_plus_z2_p	$bisim(z1,x,y) \wedge bisim(z2,x,y) \Rightarrow$ $\|z1+z2\rangle\|x\rangle p \leq \|y\rangle\|z1+z2\rangle p$	yes	0 s
fd_x_bd_z1_plus_z2_p_equ_ fd_x_bd_z1_p_plus_fd_x_bd_z2_p	$\|x\rangle\langle z1+z2\|p =$ $\|x\rangle\langle z1\|p + \|x\rangle\langle z2\|p$	misc	0 s

Table A.12: Content of `sum_bisim` (second part)

Table A.13: Content of `symm_bisim`

name	theorem	additional theorems	time
pscon_x_y_and_bisim_x_z1_z2_impl_ bisim_y_z2_z1	$pscon(x,y) \wedge bisim(x,z1,z2) \Rightarrow bisim(y,z2,z1)$	yes	0 s
pscon_x_y_and_bisim_x_z1_z2_impl_ cod_y_eq_carr_z2	$pscon(x,y) \wedge bisim(x,z1,z2) \Rightarrow \overline{\lceil y \rceil} = carr(z2)$	yes	6 s
pscon_x_y_and_bisim_x_z1_z2_impl_ dom_y_eq_carr_z1	$pscon(x,y) \wedge bisim(x,z1,z2) \Rightarrow \lceil y = carr(z1)$	yes	6 s
pscon_x_y_and_bisim_x_z1_z2_impl_ leq_bd_y_fd_z1_p_fd_z2_bd_y_p	$pscon(x,y) \wedge bisim(x,z1,z2) \wedge p \in \mathsf{test} \Rightarrow \langle y \| z1 \rangle p \leq \|z2\rangle\langle y\|p$	yes	8 s
pscon_x_y_and_bisim_x_z1_z2_impl_ leq_fd_y_fd_z2_p_fd_z1_fd_y_p	$pscon(x,y) \wedge bisim(x,z1,z2) \wedge p \in \mathsf{test} \Rightarrow \|y\rangle\|z2\rangle p \leq \|z1\rangle\|y\rangle p$	yes	17 s

name	theorem	additional theorems	time
bisim_one_top_top	$bisim(1, \top, \top)$	yes	0 s
cod_one_equ_carr_top	$\overline{1^{\urcorner}} = carr(\top)$	yes	0 s
cod_top_dot_p_equ_p	$p \in \mathsf{test} : \overline{(\top \cdot p)^{\urcorner}} = p$	misc	2 s
cod_top_equ_one	$\overline{\top^{\urcorner}} = 1$	misc	1 s
dom_one_equ_carr_top	$\lceil 1 = carr(\top)$	yes	0 s
dom_p_dot_top_equ_p	$p \in \mathsf{test} : \lceil(p \cdot \top) = p$	misc	2 s
dom_top_equ_one	$\lceil \top = 1$	misc	1 s
leq_bd_one_fd_top_p_fd_top_bd_one_p	$\langle 1 \| \top \rangle p \leq \| \top \rangle \langle 1 \| p$	yes	0 s
leq_fd_one_fd_top_p_fd_top_fd_one_p	$\| 1 \rangle \| \top \rangle p \leq \| \top \rangle \| 1 \rangle p$	yes	0 s
p_neq_zero_impl_cod_p_dot_top_equ_one	$p \in \mathsf{test} : p \neq 0 \Rightarrow \overline{(p \cdot \top)^{\urcorner}} = 1$	yes	8 s
p_neq_zero_impl_dom_top_dot_p_equ_one	$p \in \mathsf{test} : p \neq 0 \Rightarrow \lceil(\top \cdot p) = 1$	yes	8 s
top_dot_top_equ_top	$\top \cdot \top = \top$	yes	0 s
top_is_symmetric	$\top$ is symmetric	yes	9 s

Table A.14: Content of `top`

Table A.15: Content of `trans_bisim` (first part)

name	theorem	additional theorems	time
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ bisim_x_23_dot_x12_z1_z3	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow bisim(x23 \cdot x12, z1, z3)$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ carr_z2_equ_cod_x23	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow carr(z2) = x23\overline{\ }$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ cod_x23_dot_x12_equ_carr_z1	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow (x23 \cdot x12)\overline{\ } = carr(z1)$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ dom_x23_dot_carr_z2_equ_ dom_x23_dot_cod_23	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow \ulcorner(x23 \cdot carr(z2)) = \ulcorner(x23 \cdot x23\overline{\ })$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ dom_x23_dot_cod_x23_equ_carr_z3	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow \ulcorner(x23 \cdot x23\overline{\ }) = carr(z3)$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ dom_x23_dot_x12_equ_ carr_dom_x23_dot_carr_z2	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow \ulcorner(x23 \cdot x12) = carr(\ulcorner(x23) \cdot carr(z2))$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ dom_x23_dot_x12_equ_carr_z3	$bisim(x12, z1, z2) \wedge bisim(x23, z2, z3) \Rightarrow \ulcorner(x23 \cdot x12) = carr(z3)$	yes	4 s

name	theorem	additional theorems	time
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ leq_bd_x12_bd_x23_fd_z3_p_ bd_x12_fd_z2_bd_x23_p	$bisim(x12, z1, z2) \wedge$ $bisim(x23, z2, z3) \wedge p \in \mathsf{test} \Rightarrow$ $\langle x12 \vert \langle x23 \vert\vert z3 \rangle p \leq \langle x12 \vert\vert z2 \rangle \langle x23 \vert p$	yes	5 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ leq_bd_x12_fd_z2_bd_x23_p_ fd_z1_bd_x12_bd_x23_p	$bisim(x12, z1, z2) \wedge$ $bisim(x23, z2, z3) \wedge p \in \mathsf{test} \Rightarrow$ $\langle x12 \vert\vert z2 \rangle \langle x23 \vert p \leq \vert z1 \rangle \langle x12 \vert \langle x23 \vert$	yes	5 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ leq_bd_x23_dot_x12_fd_z3_p_ bd_x12_bd_x23_fd_z3_p	$bisim(x12, z1, z2) \wedge$ $bisim(x23, z2, z3) \wedge p \in \mathsf{test} \Rightarrow$ $\langle x23 \cdot x12 \vert\vert z3 \rangle p \leq \langle x12 \vert \langle x23 \vert\vert z3 \rangle p$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ leq_bd_x23_dot_x12_fd_z3_p_ fd_z1_bd_x23_dot_x12_p	$bisim(x12, z1, z2) \wedge$ $bisim(x23, z2, z3) \wedge p \in \mathsf{test} \Rightarrow$ $\langle x23 \cdot x12 \vert\vert z3 \rangle p \leq \vert z1 \rangle \langle x23 \cdot x12 \vert$	yes	0 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ leq_fd_x23_dot_x12_fd_z1_p_ fd_z3_fd_x23_dot_x12_p	$bisim(x12, z1, z2) \wedge$ $bisim(x23, z2, z3) \wedge p \in \mathsf{test} \Rightarrow$ $\vert x23 \cdot x12 \rangle \vert z1 \rangle p \leq \vert z3 \rangle \vert x23 \cdot x12 \rangle p$	yes	5 s
bisim_x12_z1_z2_and_ bisim_x23_z2_z3_impl_ leq_fd_z1_bd_x12_bd_x23_p_ fd_z1_bd_x23_dot_x12_p	$bisim(x12, z1, z2) \wedge$ $bisim(x23, z2, z3) \wedge p \in \mathsf{test} \Rightarrow$ $\vert z1 \rangle \langle x12 \vert \langle x23 \vert p \leq \vert z1 \rangle \langle x23 \cdot x12 \vert p$	yes	0 s

Table A.16: Content of `trans_bisim` (second part)

Bibliography

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Series in Computer Science and Information Processing, Addison Wesley, 1974.

[BA93] M. Ben-Ari, *Mathematical logic for computer science*, Prentice Hall, 1993.

[BAMP81] M. Ben-Ari, Z. Manna, and A. Pnueli, *The temporal logic of branching time*, Conference Record of the Eighth Annual ACM Symposium on Principles of Programming Languages — POPL 1981 (J. White, R. J. Lipton, and P. C. Goldberg, eds.), ACM Press, 1981, pp. 164–176.

[BBR04] T. Brihaye, V. Bruyère, and J.-F. Raskin, *Model-checking for weighted timed automata*, Joint International Conferences on Formal Modelling and Analysis of Timed Systems and Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems – FORMATS/FTRTFT (Y. Lakhnech and S. Yovine, eds.), Lecture Notes in Computer Science, vol. 3253, Springer, 2004, pp. 277–292.

[BC75] R. C. Backhouse and B. A. Carré, *Regular algebra applied to path-finding problems*, Journal of the Institute of Mathematics and Applications (1975).

[Bel58] R.E. Bellman, *On a routing problem*, Quart. Appl. Math. **16** (1958), 87–90.

[Ber00] D. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, 2000.

[BG09] J.N. Billings and T.G. Griffin, *A model of internet routing using semi-modules*, 11th International Conference on Relational Methods

in Computer Science — RelMiCS 2009 (R. Berghammer, A.M. Jaoua, and B. Möller, eds.), Lecture Notes in Computer Science, vol. 5827, Springer, 2009, pp. 29–43.

[BHZ77] R. E. Burkard, W. Hahn, and U. Zimmermann, *An algebraic approach to assignment problems*, Mathematical Programming (1977).

[Bir67] G. Birkhoff, *Lattice theory*, 3rd ed., Amer. Math. Soc., 1967.

[BK08] C. Baier and J-P. Katoen, *Principles of model checking*, MIT Press, 2008.

[BL69] J. R. Büchi and L. H. Landweber, *Solving sequential conditions by finite-state strategies*, Transactions of the American Mathematical Society **138** (1969), pp. 295–311.

[Bou04] A. Boulmakoul, *Generalized path-finding algorithms on semirings and the fuzzy shortest path problem*, Journal of Computational and Applied Mathematics **162** (2004), 263–272.

[BR83] S. D. Brookes and W. C. Rounds, *Behavioural Equivalence Relations Induced by Programming Logics*, 10th Colloquium on Automata, Languages and Programming (J. Díaz, ed.), Lecture Notes in Computer Science, vol. 154, Springer, 1983, pp. 97–108.

[CAD] *CADE*, `http://www.cadeinc.org/`, (accessed November 12, 2013).

[Car71] B. A. Carré, *An algebra for network routing problems*, IMA Journal of Applied Mathematics (1971).

[Car80] ———, *Graphs and networks*, Oxford Univ. Press, 1980.

[CASa] *CASC*, `http://www.cs.miami.edu/~tptp/CASC/`, (accessed November 5, 2013).

[CASb] *Results of CACS-13*, `http://www.cs.miami.edu/~tptp/CASC/13/Results.html`, (accessed November 5, 2013).

[Cas85] I. Castellani, *Bisimulations and Abstraction Homomorphisms*, Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT) (H. Ehrig, C. Floyd, M. Nivat, and J. W. Thatcher, eds.), Lecture Notes in Computer Science, vol. 185, Springer, 1985, pp. 223–238.

[CGK+13] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse, *An overview of the mcrl2 toolset and its recent advances*, Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference – TACAS 2013 (N. Piterman and S. A. Smolka, eds.), Lecture Notes in Computer Science, vol. 7795, Springer, 2013, pp. 199–213.

[CGP01] E. Clarke, O. Grumberg, and D. Peled, *Model checking*, 3rd ed., MIT Press, 2001.

[Coh00] E. Cohen, *Separation and reduction*, Mathematics of Program Construction — MPC 2000 (R. C. Backhouse and J. N. Oliveira, eds.), Lecture Notes in Computer Science, vol. 1837, Springer, 2000, pp. 45–59.

[Con] A. Condon, *On algorithms for simple stochastic games*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **13**.

[Con92] ———, *The complexity of stochastic games*, Information and Computation, vol. 96, 1992, pp. 203–224.

[DBL] *DBLP*, `http://dblp.uni-trier.de/`, (accessed August 16, 2013).

[Der72] C. Derman, *Finite State Markov Decision processes*, Academic Press, 1972.

[DGW13] D. Delling, A. V. Goldberg, and R. F. Werneck, *Hub label compression*, Experimental Algorithms, 12th International Symposium — SEA 2013, Lecture Notes in Computer Science, vol. 7933, Springer, 2013, pp. 18–29.

[Dij] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1**.

[DMS06] J. Desharnais, B. Möller, and G. Struth, *Kleene algebra with domain*, ACM Transactions on Computational Logic **7** (2006), 798–833.

[DMS11] J. Desharnais, B. Möller, and G. Struth, *Algebraic notions of termination*, Logical Methods in Computer Science **7** (2011), no. 1.

[DRDW06] L. Doyen, J.-F. Raskin, and M. De Wulf, *A lattice theory for solving games of imperfect information*, 9th International Workshop on Hybrid Systems: Computation and Control – HSCC 2006 (J. P. Hespanha and A. Tiwari, eds.), Lecture Notes in Computer Science, vol. 3927, Springer, 2006, pp. 153–168.

[DW13] D. Delling and R. F. Werneck, *Faster customization of road networks*, Experimental Algorithms, 12th International Symposium — SEA 2013, Lecture Notes in Computer Science, vol. 7933, Springer, 2013, pp. 30–42.

[Ehm03] Thorsten Ehm, *Pointer kleene algebra*, Relational and Kleene-Algebraic Methods in Computer Science: 7th International Seminar on Relational Methods in Computer Science and 2nd International Workshop on Applications of Kleene Algebra – RelMiCS 2003 (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 3051, Springer, 2003, pp. 99–111.

[EKL08] J. Esparza, S. Kiefer, and M. Luttenberger, *Derivation tree analysis for accelerated fixed-point computation*, Developments in Language Theory (I. Masami and T. Masafumi, eds.), Lecture Notes in Computer Science, vol. 5257, Springer, 2008, pp. 301–313.

[EKMS94] M. Erné, J. Koslowski, A. Melton, and G. Strecker, *A primer on galois connections*, Papers on general topology and its applications — 7th Summer Conf. Wisconsin (S. et al Andima, ed.), Annals New York Accd. Sci., vol. 704, 1994, pp. 103–125.

[EMS03a] T. Ehm, B. Möller, and G. Struth, *Kleene modules*, Relational and Kleene-Algebraic Methods in Computer Science: 7th International Seminar on Relational Methods in Computer Science and 2nd International Workshop on Applications of Kleene Algebra – RelMiCS 2003 (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 3051, Springer, 2003, pp. 112–124.

[EMS03b] ———, *Kleene modules*, Tech. Report 2003-10, Institut für Informatik, Universität Augsburg, 2003.

[Fed80] A. Federgruen, *Successive approximation methods in undiscounted stochastic games*, Operations Research **28** (1980), no. 3, 794–809.

[Fer90] J.-C. Fernandez, *An implementation of an efficient algorithm for bisimulation equivalence*, Science of Computer Programming (1990), 219–236.

[FN08] S. Z. Fazekas and B. Nagy, *Scattered Subword Complexity of non-primitive Words*, Journal of Automata, Languages and Combinatorics **13** (2008), no. 3, 233–247.

[FSTV91] J. A. Filar, T. A. Schultz, F. Thuijsman, and O. J. Vrieze, *Nonlinear programming and stationary equilibria in stochastic games*, Mathematical Programming **50** (1991), 227–237.

[FUMK06] H. Foster, S. Uchitel, J. Magee, and J. Kramer, *Ltsa-ws: a tool for model-based verification of web service compositions and choreography*, 28th International Conference on Software Engineering – ICSE 2006 (L. J. Osterweil, H. D. Rombach, and M. L. Soffa, eds.), ACM, 2006, pp. 771–774.

[Glüa] R. Glück, *Personal Homepage (English)*, `https://www.informatik.uni-augsburg.de/en/chairs/dbis/pmi/staff/glueck/PhD/`, (accessed November 25, 2013).

[Glüb] ———, *Personal Homepage (German)*, `https://www.informatik.uni-augsburg.de/lehrstuehle/dbis/pmi/staff/glueck/PhD/`, (accessed November 25, 2013).

[Glü11] ———, *Using bisimulations for optimality problems in model refinement*, 12th International Conference on Relational and Algebraic Methods in Computer Science — RAMICS 2011 (H. de Swart, ed.), Lecture Notes in Computer Science, vol. 6663, Springer, 2011, pp. 164–179.

[Glü12] ———, *Two observations in dioid based model refinement*, 13th International Conference on Relational and Algebraic Methods in Computer Science — RAMICS 2012 (T. G. Griffin and W. Kahl, eds.), Lecture Notes in Computer Science, vol. 7560, Springer, 2012, pp. 235–247.

[GM08a] R. Glück and B. Möller, *Circulations, fuzzy relations and semirings*, Mathematics of Program Construction — MPC 2008 (P. Audebaud and C. Paulin-Mohring, eds.), Lecture Notes in Computer Science, vol. 5133, Springer, 2008, pp. 134–152.

[GM08b] M. Gondran and M. Minoux, *Graphs, dioids and semirings*, Springer, 2008.

[GMS09] R. Glück, B. Möller, and M. Sintzoff, *A semiring approach to equivalences, bisimulations and control*, 11th International Conference on Relational Methods in Computer Science — RelMiCS 2009 (R. Berghammer, A.M. Jaoua, and B. Möller, eds.), Lecture Notes in Computer Science, vol. 5827, Springer, 2009, pp. 134–149.

[GMS11] ———, *Model refinement using bisimulation quotients*, 13th International Conference on Algebraic Methodology And Software Technology — AMAST 2010 (M. Johnson and D. Pavlovic, eds.), Lecture Notes in Computer Science, vol. 6486, Springer, 2011, pp. 76–91.

[GMvWU06] J. F. Groote, A. Mathijssen, M. van Weerdenburg, and Y. S. Usenko, *From μCRL to mCRL2*, Electronic Notes in Theoretical Computer Science **162** (2006), 191–196.

[HHW95] M. R. Henzinger, T. A. Henzinger, and Kopke. P. W., *Computing simulations on finite and infinite graphs*, FOCS, IEEE Computer Society Press, 1995, pp. 453–462.

[Hig52] G. Higman, *Ordering by Divisibility in Abstract Algebras*, Proceedings of the London Mathematical Society **s3-2** (1952), no. 1, 326–336.

[HK66] A. J. Hoffman and R. M. Karp, *On nonterminating stochastic games*, Management Science **12** (1966), no. 5, 359–370.

[HMCJF00] T. A. Henzinger, R. Majumdar, Mang F. Y. C., and Raskin J.-F., *Abstract interpretation of game properties*, 7th International Symposium on Static Analysis – SAS 2000 (J. Palsberg, ed.), Lecture Notes in Computer Science, vol. 1824, Springer, 2000, pp. 220–239.

[Höf09] Peter Höfner, *Algebraic calculi for hybrid systems*, Ph.D. thesis, 2009.

[How66] R. A. Howard, *Dynamic programming and markov processes*, M.I.T. Press, 1966.

[HS07] P. Höfner and G. Struth, *Automated reasoning in Kleene algebra*, Automated Deduction — CADE–21 (F. Pfennig, ed.), Lecture Notes in Artificial Intelligence, vol. 4603, Springer, 2007, pp. 279–294.

[HS08] Peter Höfner and Georg Struth, *On automating the calculus of relations*, 4th International Joint Conference on Automated Reasoning — IJCAR–2008 (Alessandro Armando, Peter Baumgartner, and Gilles Dowek, eds.), Lecture Notes in Computer Science, vol. 5195, Springer, 2008, pp. 50–66.

[IJC] *IJCAR*, `http://www.ijcar.org/`, (accessed November 12, 2013).

[Inta] *Intel 4004 Circuit*, `http://www.intel.com/Assets/PDF/General/4004_schematic.pdf`, (accessed November 25, 2013).

[Intb] *Intel Homepage*, `http://www.intel.com`, (accessed November 25, 2013).

[JKM98] P. Jančar, A. Kučera, and R. Mayr, *Deciding bisimulation-like equivalences with finite-state processes*, 25th International Colloquium on Automata, Languages and Programming – ICALP'98 (K. G. Larsen, S. Skyum, and G. Winskel, eds.), Lecture Notes in Computer Science, vol. 1443, Springer, 1998, pp. 200–211.

[Jun05] Dieter Jungnickel, *Graphs, networks and algorithms*, 2nd ed., Springer Verlag, 2005.

[Kaw06] Yasuo Kawahara, *On the cardinality of relations*, RelMiCS, 2006, pp. 251–265.

[Kle52] S.C. Kleene, *Introduction to metamathematics*, Wolters-Noordhoff - Groningen, 1952.

[Koz90] D. Kozen, *On kleene algebras and closed semirings*, Mathematical Foundations of Computer Science 1990 — MFCS 1990 (B. Rovan, ed.), Lecture Notes in Computer Science, vol. 452, Springer, 1990, pp. 26–47.

[Koz94] _____, *A completeness theorem for kleene algebras and the algebra of regular events*, Information and Computation **110** (1994), no. 2, 366–390.

[KPP09] H. Kugler, C. Plock, and A. Pnueli, *Controller synthesis from lsc requirements*, 12th International Conference on Fundamental Approaches to Software Engineering, FASE 2009 (M. Chechik and M. Wirsing, eds.), Lecture Notes in Computer Science, vol. 5503, Springer, 2009, pp. 79–93.

[KS90] P. C. Kanellakis and S. A. Smolka, *CCS Expressions, Finite State Processes, and Three Problems of Equivalence*, Information and Computation **86** (1990), no. 1, 43–68.

[Kya66] L.G. Kyachiyan, *A polynomial time algorithm for linear programming*, Soviet Math Dokl. **20** (1966), 359–370.

[LTS] *LTSA*, `http://www.doc.ic.ac.uk/ltsa/`, (accessed August 12, 2013).

[MB85] E. Manes and D. Benson, *The inverse semigroup of a sum-ordered semiring*, Semigroup Forum **31** (1985), 129–152.

[MC94] M. Melekopoglou and A. Condon, *On the complexity of the policy improvement algorithm for markov decision processes*, INFORMS Journal on Computing **6** (1994), no. 2, 188–192.

[McC] W. W. McCune, *Prover9 and Mace4*, `http://www.cs.unm.edu/~mccune/prover9`, (accessed April 18, 2013).

[mCL] *mCLR2*, `http://www.mcrl2.org`, (accessed August 12, 2013).

[Mil80] R. Milner, *A calculus of communicating systems*, Lecture Notes in Computer Science, vol. 92, Springer, 1980.

[Min76] M. Minoux, *Structures algébriques généralisées des problèmes de cheminement dans les graphes: Théorèmes, algorithmes et applications*, R.A.I.R.O. Recherche Opérationnelle (1976).

[MM79] G. Milne and R. Milner, *Concurrent processes and their syntax*, Journal of the ACM **26** (1979), no. 2, 302–321.

[Möl13] B. Möller, *Modal knowledge and game semirings*, Computer Journal **56** (2013), no. 1, 53–69.

[MP82] Z. Manna and A. Pnueli, *Verification of concurrent programs: Temporal proof principles*, Logic of Programs, Workshop, Yorktown Heights, New York, May 1981 (D. Kozen, ed.), Lecture Notes in Computer Science, vol. 131, Springer, 1982, pp. 200–252.

[MP84] ———, *Adequate proof principles for invariance and liveness properties of concurrent programs*, Science of Computer Programming **4** (1984), no. 3, 257–289.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis, *On the synthesis of discrete controllers for timed systems (an extended abstract)*, 12th Annual Symposium on Theoretical Aspects of Computer Science — STACS 95 (E. W. Mayr and C. Puech, eds.), Lecture Notes in Computer Science, vol. 900, Springer, 1995, pp. 229–242.

[Myh57] J. Myhill, *Finite automata and the representation of events*, WADD TR-57-624 (1957), 112–137.

[Ner58] A. Nerode, *Linear automaton transformations*, Proceedings of the American Mathematical Society, vol. 9, 1958, pp. 541–544.

[Plo76] G. D. Plotkin, *A powerdomain construction*, SIAM Journal on Computing **5** (1976), no. 3, 452–487.

[Pnu77] Amir Pnueli, *The temporal logic of programs*, 18th Annual Symposium on Foundations of Computer Science — FOCS18, IEEE Computer Society, 1977, pp. 46–57.

[Pou07] D. Pous, *Complete lattices and up-to techniques*, Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007 (S. Zhong, ed.), Lecture Notes in Computer Science, vol. 4807, Springer, 2007, pp. 351–366.

[PT] R. Paige and R. Tarjan, *Three partition refinement algorithms*, SIAM Journal on Computing **16(6)**.

[PV] A. Puhakka and A. Valmari, *Liveness and fairness in process-algebraic verification*, Proceedings of the 12th International Conference on Concurrency Theory — CONCUR 2001.

[RW89] P. J. Ramadge and W. M. Wonham, *The control of discrete event systems*, Proceedings of the IEEE 77, 1989, pp. 81–98.

[Sal03] A. Salomaa, *Counting (scattered) Subwords*, Bulletin of the EATCS **81** (2003), 165–179.

[Sha53] L.S. Shapley, *Stochastic games*, Proceedings of the National Academy of Sciences, vol. 39, 1953, pp. 1095–1100.

[Son90] E. D. Sontag, *Mathematical Control Theory*, Springer, 1990.

[SS93] G. Schmidt and T. Ströhlein, *Relations and graphs: Discrete mathematics for computer scientists*, Springer, 1993.

[Tar55] A. Tarski, *A lattice-theoretical fixpoint theorem and its applications*, Pacific Journal of Mathematics **5** (1955), no. 2, 285–309.

[TW91] J. G. Thistle and W. M. Wonham, *Control of omega-automata, church's problem, and the emptiness problem for tree omega-automata*, Computer Science Logic, 5th Workshop — CSL5 (E. Egon Börger, G. Jäger, H. K. Büning, and M. M. Richter, eds.), Lecture Notes in Computer Science, vol. 626, Springer, 1991, pp. 367–382.

[TW94] ———, *Control of infinite behavior of finite automata*, SIAM Journal on Control and Optimization **Volume 32** (1994), no. 4, 1075–1097.

[UCKM03] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, *Ltsa-msc: Tool support for behaviour model elaboration using implied scenarios*, Tools and Algorithms for the Construction and Analysis of Systems - 9th International Conference – TACAS 2003 (H. Garavel and J. Hatcliff, eds.), Lecture Notes in Computer Science, vol. 2619, Springer, 2003, pp. 597–601.

[vdW76] J. van der Wal, *A successive approximation algorithm for an undiscounted markov decision process*, Computing **17** (1976), no. 2, 157–162.

[Vin00] R. Vinter, *Optimal control*, Birkhäuser, 2000.

[Win08] M. Winter, *A relation-algebraic theory of bisimulations*, Fundam. Inf. **83** (2008), no. 4, 429–449.

List of Figures

List of Tables

List of Algorithms

Index

Wenn zwei das Gleiche tun, ist es noch lange nicht dasselbe.

German Proverb

Curriculum Vitae

Roland Glück
glueck@informatik.uni-augsburg.de

Date of Birth	25 August 1972
Citizenship	German
Education	
2007	Diploma in Computer Science (major) and Mathematics (minor) at University of Augsburg
1991 - 1993	Study of Mathematics at University of Augsburg
1991	Abitur
Academic Positions	
2007 - 2013	Researcher at University of Augsburg
Non-Academic Positions	
2002 - 2007	Demonstrator at University of Augsburg