NOTE TO USERS

This reproduction is the best copy available.

UMI
FINITE-STATE MODELS WITH MULTIPLICITIES: SYMBOLIC REPRESENTATION AND REASONING

by

Benet Devereux

A thesis submitted in conformity with the requirements for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Copyright © 2001 by Benet Devereux
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.
Abstract

Finite-State Models with Multiplicities: Symbolic Representation and Reasoning

Benet Devereux

Master of Science

Graduate Department of Computer Science

University of Toronto

2001

It is useful to add to the descriptiveness of finite-state models by placing weights, or multiplicities, on the transition. There is considerable ad hoc work in the formal verification community on temporal logics for properties of such systems, largely for probabilistic multiplicities; also, there is a branch of automata theory dealing with transition multiplicities in a very general algebraic setting, but it has not, to date, been concerned with verification.

This thesis attempts to bridge this gap by defining Kripke structures with transition multiplicities, and three variants of branching-time temporal calculus for stating computations over such transition systems. We present two decision-diagram varieties which may be useful for compact symbolic representation and reasoning in such systems: these varieties, shared DDs and edge-valued DDs, are generalizations to arbitrary algebras of structures already defined for numeric functions. Finally, we offer a simple symbolic model-checking algorithm for one of the temporal calculi.
## Contents

1 Introduction ................................................................. 6  
1.1 Background ................................................................ 6  
1.2 Contributions of this Thesis ........................................ 7  
1.3 Related Work ................................................................ 9  
1.4 Organization of this Thesis ............................................ 9  

2 Mathematical Background .................................................. 11  
2.1 Binary Operations ....................................................... 11  
2.2 Digits ........................................................................ 12  
2.3 Partial Orders and Lattices ............................................ 12  
2.4 Sets .......................................................................... 13  
   2.4.1 Standard Sets ...................................................... 13  
   2.4.2 Set Constructions: Products and Powers ................ 13  
   2.4.3 Sets with Multiplicities ....................................... 14  
2.5 Relations with Multiplicities .......................................... 16  
   2.5.1 Basic Definitions ................................................ 16  
   2.5.2 Composition and Image ....................................... 17  
2.6 Semirings ................................................................... 19  
2.7 Additional Structure .................................................... 22  
   2.7.1 Optimistic and Pessimistic Choice ....................... 22
CONTENTS

2.7.2 Implication .............................................. 23
2.7.3 Transition Algebras ...................................... 24

3 Finite-State Models with Multiplicities .............. 26
  3.1 Background .................................................. 26
  3.2 Kripke Structures with Transition Multiplicities .......... 26
  3.3 Multiplicities for Modelling .............................. 28
    3.3.1 Examples of Transition Algebras ..................... 28
    3.3.2 Construction of Transition Algebras .................. 30
    3.3.3 An Example of Transition Engineering ................. 31

4 Propositional Temporal Calculi ......................... 34
  4.1 Background .................................................. 34
  4.2 Temporal Logic ............................................. 34
  4.3 From Logic to Calculus .................................. 35
    4.3.1 State, Transition, and Path Values .................. 35
    4.3.2 Path Quantification .................................. 36
  4.4 Three Temporal Calculi .................................. 38
    4.4.1 Purposes ................................................. 38
    4.4.2 KCTL\textsubscript{1}*: Valued Paths, Boolean States ........ 38
    4.4.3 KCTL\textsubscript{2}*: Valued Paths and States, Optimism and Pessimism ... 41
    4.4.4 KCTL\textsubscript{3}*: Valued Paths and States, Implication ........... 43
    4.4.5 Valued State Variables ................................. 44
    4.4.6 Multiple-Valued Model-Checking ....................... 45
  4.5 Evaluation of Formulae .................................. 46

5 Decision Diagrams ........................................... 48
  5.1 Definitions ............................................... 48
    5.1.1 Decomposing Finite Functions ......................... 49
CONTENTS

5.1.2 Syntax .................................................. 51
5.1.3 Semantics ................................................ 52
5.1.4 Representing State Sets and Transition Relations .......... 53
5.2 Decision Diagram API and Implementation .................... 53
5.3 Decision Diagram Varieties ................................ 55
  5.3.1 Shared Decision Diagrams .............................. 55
  5.3.2 Binary Representations ............................... 57
  5.3.3 Edge-Valued Decision Diagrams ....................... 58
  5.3.4 Example ............................................. 61

6 Symbolic Reachability Analysis with Multiplicities .......... 66
  6.1 Reachability Analysis ..................................... 66
  6.2 Convergence and Approximation ........................... 68
  6.3 Symbolic Model-Checking KCTL_3 ......................... 69

7 Conclusions and Future Work .................................. 73
  7.1 Conclusions ............................................. 73
  7.2 Future Work ............................................ 74
  7.3 Acknowledgements ....................................... 74

A Decision Diagram Operations ................................. 76
  A.1 Shared BDDs ............................................ 76
  A.2 Normalized Edge-Valued DDs .............................. 78
Chapter 1

Introduction

1.1 Background

State-based models are a natural way to describe many systems, both real-world and engineered. They are straightforward to create and communicate, with an intuitive graphical representation, and yet are amenable to thorough formal analysis, through technologies such as model-checking [17]. Even though finite-state models can rapidly grow to a large size through parallel composition or introduction of numeric variables, their analysis is kept computationally tractable in practice through symbolic representations [38, 12].

Basic to any state-based model of a system is its transition relation or accessibility relation: the set of assertions that, for any two possible system states, it is either possible to reach the second from the first in a single step, or it is not. For the purpose of more informative modelling, the modeller may desire to say something about this step: what is the probability that it is taken? How certain are we that it is even there to be taken? What does it cost, in terms of time or some other limited resource, to take it? What input needs to be received from the environment to enable it, or, conversely, what output is emitted as it is taken?

These values attached to transitions are often called multiplicities. There is a sub-
stantial body of work dealing with terminating transition systems with multiplicities [21, 44, 32, 8], but less dealing with those that model infinite computations.

There are a number of reasons for attaching multiplicities to transitions:

- Components of the system have empirically or theoretically determined probabilities of failing or at least malfunctioning. In this case, probabilistic model-checking [6] can determine whether the probability that the whole system fails to behave properly is sufficiently small.

- Transitions between states take a fixed amount of time, as in timed event graphs, or possibly require fixed amounts of several different resources [24].

- The system model is built from several viewpoints which may conflict, and transition multiplicities reflect the confidence of stakeholders in the possibility of a particular change of state [15, 14].

Richer multiplicities can be constructed from simpler ones; probability and cost, for instance, can be combined.

In general, transition multiplicities arise from the need to formally express requirements that have multiplicities themselves, rather than being on/off conditions: such as fault-tolerance, quality of service, and satisfying a majority of stakeholders. The principal purpose of the current work is to propose a unified foundation for modelling and reasoning using transition multiplicities, and also for the task of choosing such multiplicities based on analysis goals: this task might be dubbed transition engineering.

1.2 Contributions of this Thesis

In the background section, we give a point-by-point argument from intuition for the necessity of each of the semiring axioms in defining transition multiplicities. This argument is new, as far as we know, and offers an exposition where mostly we have only
found definitions.

The following chapters - on Kripke structures with multiplicity, and temporal calculi over them - are a step towards a synthesis of two areas of computer science that have not, as it seems, communicated much: the very theoretical branch of formal power series, which offers a rich set of results on finite automata and finite-length languages with multiplicities, and the more applied area of formal verification. It is the author's belief that there are many powerful insights in the formal power series literature which can be profitably applied in the realm of verification and system analysis, and this thesis is meant as a proof of that concept.

More specifically, the bridge this thesis endeavours to construct is the enrichment of branching-time temporal logic [38] by treating the algebra of transitions as a degree of freedom. We argue that this treatment unifies several ad hoc advances in the verification literature.

The distinction made by branching-time temporal logic between “sometimes” and “always” led to two necessary enhancements to transition algebras: addition of a pessimistic choice operator (as well as the ordinary optimistic choice), and the generalization of implication.

The perspective of the algebra of transitions as a degree of freedom is also applied to decision diagrams, the standard symbolic representation. We propose a more algebraic characterization of the shared decision diagrams of Minato et al. [39] and also of the factored edge-valued decision diagrams of Tafertshofer et al. [47]. It is our hope that this view may be of some use when these technologies are being applied to new domains of interest in the formal verification community.
1.3 Related Work

Since probabilistic transition systems – Markov chains – have been around for many years, there have been a variety of approaches taken to temporal logics for probabilistic systems, starting with the probabilistic branching-time logic PCTL of Hansson and Jonsson [27]. Several ideas from this work – such as thresholds in path quantification – have been reused here, and we have attempted to restate them in the more general setting.

Baier and Clarke [5] have generalized the propositional $\mu$-calculus of Kozen [31] to produce the “algebraic $\mu$-calculus”, a language for expressing fixed-point computations involving real-valued functions. These include shortest-path problems, and asymptotic behaviour of Markov chains.

Godefroid and Bruns [10] have defined a 3-valued CTL. Their work, like this thesis, makes the distinction between optimistic and pessimistic choice, although in a somewhat different manner.

More in the domain of philosophical logic, Fitting [22] has developed a many-valued modal logic, where the accessibility relation values come from a Heyting algebra. This logic is linear-time, by contrast with the branching-time calculi proposed here.

In the body of the thesis, the particular relations between the current work and this related work will be drawn at the relevant points.

1.4 Organization of this Thesis

The thesis begins with a statement of some mathematical preliminaries. In Chapter 2, we introduce all of the necessary notations and theory to develop the material. The most important concept is viewing functions assigning values to elements of some domain as “sets” and “relations” with multiplicities, with rules for intersection, union, complementation, and relational product; some algebraic structure underlying ranges of valuation is introduced.
In Chapter 3 this algebraic structure is used to define transition systems (Kripke structures) with values attached to their transitions, and to give values to executions of such systems. Some specific examples of transition algebras that have been used are presented, in order to help build intuition about the more general framework.

Chapter 4 defines three variant languages for specifying computations in the Kripke structures with multiplicities defined in Chapter 3 – these are called "temporal calculi", and considered as an extension of the branching-time temporal logic CTL*.

In Chapter 5, we move from theory to implementation, and discuss decision diagrams, a standard data structure for representing transition systems and computing values of temporal-logic formulae using symbolic model-checking. The fundamental ideas of decision diagrams are reviewed. We present two extensions of decision diagrams, shared DDs and edge-valued DDs, which may, for some transition algebras, provide more efficient representation and manipulation than standard decision diagrams. Algorithms for creating and using these structures are given in the Appendix.

Chapter 6 shows how the decision diagrams of Chapter 5 can be used to evaluate a subset of the formulae from the temporal calculi of Chapter 4.

The final chapter is the conclusion, and sketches some possibilities for future work expanding upon what is described in the thesis.
Chapter 2

Mathematical Background

In this chapter we present the notation used throughout the thesis, and some useful definitions. None of the notation used is completely novel, although some of it (such as the infix forms of maximum and minimum) is not entirely standard; all notational choices have been made for the sake of compactness and avoidance of ambiguity. At times this may come at the expense of familiarity.

2.1 Binary Operations

The symbols $\oplus$ and $\otimes$ denote arbitrary binary operations. The following binary operations are used throughout:

- $\oplus, \otimes$: standard addition and multiplication of numbers.

- $\uparrow, \downarrow$: maximum or minimum relative to a specified total order.

All binary operations are written infix, including $\uparrow$ and $\downarrow$. 
2.2 Digits

We denote the set of \( n \)-ary digits by \([n]\): that is, the set of binary digits 0, 1 is \([2]\), the set of quaternary digits 0, 1, 2, 3 is \([4]\), and so on.

2.3 Partial Orders and Lattices

A partial order is a relation \( \subseteq \) - read “under” - which is reflexive, transitive, and antisymmetric. Given a subset \( A \) of the domain of the relation, the \textit{infimum} of \( A \), if it exists, is the greatest element in the domain which is below every element of \( A \): it is written \( \inf A \). The \textit{supremum} is the least element above every element of \( A \), written \( \sup A \). Note that \( \inf A \) and \( \sup A \) need not actually be contained in \( A \).

More formally, for all \( a \in A \):

\[
\inf A \subseteq a \\
\quad a \subseteq \sup A
\]

A \textit{lattice} is a structure that can be defined in one of two ways:

- A pair \((L, \subseteq)\), comprising a carrier set \( L \) and a partial order \( \subseteq \), such that for all finite \( A \subseteq L \), both \( \inf A \) and \( \sup A \) exist and are in \( L \).

- A triple \((L, \cap, \cup)\), comprising a carrier set \( L \) and with two operations \( \cap \) and \( \cup \) - pronounced “meet” and “join” - that are associative, commutative, and idempotent.

The definitions are complementary, and linked by the following axioms, which hold for all \( \ell, \ell' \in L \):

\[
(\ell \cap \ell') = \sup\{\ell'' \mid \ell'' \subseteq \ell \land \ell'' \subseteq \ell'\} \quad \text{(Greatest Lower Bound)}
\]

\[
(\ell \cup \ell') = \inf\{\ell'' \mid \ell \subseteq \ell'' \land \ell \subseteq \ell'\} \quad \text{(Least Upper Bound)}
\]
A complete lattice $\mathcal{L}$ contains two distinguished elements, $T$ and $\bot$ (read “top” and “bottom”), such that $T = \sup \mathcal{L}$ and $\bot = \inf \mathcal{L}$. All finite lattices are also complete.

A lattice is distributive if $\cap$ distributes over $\cup$ and $\cup$ over $\cap$. It is quasi-boolean if an operation $\neg$, called a quasi-complement, can be defined such that for all $\ell, \ell' \in \mathcal{L}$:

$$\ell \subseteq \ell' \Leftrightarrow \neg \ell' \subseteq \neg \ell \quad \text{(Antimonotonicity)}$$

$$\neg \neg \ell = \ell \quad \text{(Involution)}$$

$$\neg (\ell \cap \ell') = \neg \ell \cup \neg \ell' \quad \text{(De Morgan 1)}$$

$$\neg (\ell \cup \ell') = \neg \ell \cap \neg \ell' \quad \text{(De Morgan 2)}$$

In a quasi-boolean lattice, the generalized material implication or relative pseudo-complement is defined as follows:

$$\ell \rightarrow \ell' \equiv \neg \ell \cup \ell'$$

We refer to [19] as a standard reference on partial orders and lattices.

2.4 Sets

2.4.1 Standard Sets

We frequently refer to: the integers (positive, negative, and zero) $\mathbb{Z}$, the natural numbers $\mathbb{N} = \{n \mid n \in \mathbb{Z}, n \geq 0\}$, the real numbers $\mathbb{R}$, the closed interval $I = \{x \mid x \in \mathbb{R}, 0 \leq x \leq 1\}$, and the set $\mathcal{B} = \{\top, \bot\}$ of classical truth values. (Note that this is also a lattice, with $\bot \subseteq \top$.)

2.4.2 Set Constructions: Products and Powers

The Cartesian product of two sets, $S \times T$, is defined in the usual manner: \{(s, t) \mid s \in S \land t \in T\}. The set $S^T$ - “$S$ to the power of $T$” consists of all of the total functions
from $T$ into $S$; $\{2\}^S$ is the set of all subsets of $S$. This set is isomorphic to $\mathbb{B}^S$, the set of characteristic functions or membership predicates of subsets of $S$. The power $S^{[n]}$ – more conventionally written just $S^n$ – is the $n$th Cartesian power of $S$. Its elements can be considered either as tuples $(s_0, s_1, \ldots, s_{n-1})$ or as vectors: in which case they are denoted with an arrow above the name, as in $\vec{v}$.

Individual elements of vectors are referenced by removing the arrow and subscripting. For instance, let $\vec{v} \in \{2\}^3$ be $(0, 1, 0)$. Then $v_1 = 0$, $v_2 = 1$. Concatenation of vectors is indicated by apposition: if $\vec{v} \in \{2\}^2$ is equal to $(0, 1)$, then $\vec{v} \vec{w} \in \{2\}^5$ is $(0, 1, 0, 0, 1)$.

### 2.4.3 Sets with Multiplicities

We can extend the concept of a set with additional information. In ordinary sets, we say that some candidate for set membership is either in or out of the set. Thus, membership is decided by a predicate or characteristic function: a function from the universe of discourse to the range $\mathbb{B}$. If this predicate is allowed to have a different range, with some formal definition of intersection and union, then we obtain a function over the universe of discourse, which can be thought of as a set with multiplicity.

We formalize this notion as follows, using the definition of powers of sets from the previous section.

**Definition 1** Let $S$, $T$ be sets. Then a total function $\varphi \in S^T$ is called a subset of $T$ with multiplicities in $S$, or a valuation of $T$ (in $S$). For any $t \in T$, $\varphi(t)$ is called the membership degree of $t$ in $\varphi$. Functions from $S$ to itself are lifted to $S^T$ pointwise: that is,

$$f^\varphi(\varphi) = \lambda t \in T. f(\varphi(t))$$

for unary functions, and

$$f^\varphi(\varphi, \psi) = \lambda t \in T. \lambda t' \in T. f(\varphi(t), \psi(t))$$

for binary functions; the generalization to ternary or greater is clear.
Some examples follow:

**Example 1** If $\varphi$ assigns to every element of $S$ a natural number, $\varphi$ is called a multiset or bag on $S$: it is an element of $\mathbb{N}^S$. We perform "union" of multisets by lifted addition: the intuition here is that we take all of the elements from individual multisets and put them together. We perform intersection by taking the minimum. One multiset $A$ is said to be a sub-multiset of $B$, if they have the same domain $S$, and for all $s \in S$,

$$A(s) \leq B(s)$$

For any ordered or partially-ordered $T$, the subset relation can be generalized in this manner: this simply amounts to lifting the partial-order relation to $T$-valued functions.

**Example 2** [48] Fuzzy subsets of $S$ are elements of $\mathcal{I}^S$. Union of fuzzy sets is taken by pointwise maximum of membership degrees; intersection, by pointwise minimum. If instead of $\mathcal{I}$, an arbitrary partial order, such as the set of values of a multiple-valued logic [9], is used, we obtain multiple-valued sets [14], which have also been called $L$-fuzzy sets [25].

In the conventional algebra of sets, we have also the operation of set complementation, which we will also want to generalize. Fuzzy subsets, for instance, have an obvious complementation: subtract the membership degree of each element from 1. This has all of the properties that we usually expect from set complementation: it is an involution $(1 - (1 - x) = x)$, and De Morgan's laws remain valid. Furthermore, it is antimonotonic: if $x < y$, then $(1 - y) < (1 - x)$.

No such tractable complement presents itself for multisets: the most obvious candidate is the function which takes any non-zero number onto zero, and zero onto 1. This behaves somewhat like our intuition of a complement: if an element has at least one instance in the multiset, then it is removed entirely; and if some element of the universe has no instances in the multiset, then exactly one is added. It is thus antimonotonic, but
not an involution: performing it twice does not return our original multiset. Furthermore, it does not completely obey the usual De Morgan laws with respect to negation; see Section 2.3. However, it seems like "enough" of a complement to satisfy our intuitive notion.

Since our goal is the modelling of dynamic behaviour, we also require relations with multiplicities, as well as sets.

### 2.5 Relations with Multiplicities

#### 2.5.1 Basic Definitions

**Definition 2** Let $S, T, \Sigma$ be sets. A total function $R \in \Sigma^{S \times T}$ is a relation with multiplicities in $\Sigma$: we call $R(s, t)$ the relatedness of $s$ and $t$.

**Example 3** If $\Sigma = \{0, 1\}$, the relation is a standard one: two elements are either related or not.

We want to be able to compose two relations. In order to extend the ordinary definition of sequential composition, we need to add a certain amount of algebraic structure to the set of multiplicities. Given the task at hand, it is perhaps best to express the need for this structure in terms of valuations of paths through a transition relation.

In Figure 2.1, we see two situations. In (a), there is one path connecting states $s_i$ and $s_f$ via an intermediate state $s_a$. The transition from $s_i$ has the value $\alpha$, and the transition from $s_a$ to $s_f$ has the value $\beta$: we need some operation on multiplicities which yields the value of this path in terms of $\alpha$ and $\beta$.

In Figure 2.1(b), there are multiple paths between $s_i$ and $s_f$, each with a different value associated with it. We also need an operation which gives us the "overall" value of the path from $s_i$ to $s_f$: this operation may yield information as simple as "yes, there is a path", or as rich as "this set of input words moves the system from $s_i$ to $s_f".
2.5.2 Composition and Image

**Definition 3** A sequencing operation on values is an associative binary operator $\otimes : \Sigma \times \Sigma \rightarrow \Sigma$, with a unique identity $1_{\Sigma}$. In other words, $(\Sigma, \otimes, 1_{\Sigma})$ is a monoid.

**Example 4** Let $\Sigma = \mathbb{I}$ and $\mathcal{R} \in \Sigma^{S \times T}$. If, for every $s \in S$, $\sum_{t \in T} \mathcal{R}(s, t) = 1$, then $\mathcal{R}$ can be considered a probabilistic or stochastic relation (under certain additional constraints which are not relevant here [41]). If we consider a situation such as that of Figure 2.1(a), we intuit that the value on the path from $s_i$ to $s_f$ should be equal to $\alpha \times \beta$, the probability that $s_a$ is reached from $s_i$ and $s_f$ is reached from $s_a$. $\times$ is the sequencing operation, with identity $1$.

We now consider the second question: what if, in a sequential composition, there is more than one possible value for $\mathcal{R} \cdot \mathcal{R}(s, u)$? We feel that, unlike the case with the sequencing relation, it should not matter in what order we take the possible values when computing the overall value: so this operation should commute. There should, however, be an identity, a value which never influences the computation.

**Definition 4** A choice operation on values is an associative binary operator $\oplus : \Sigma \times \Sigma \rightarrow \Sigma$, with a unique identity $0_{\Sigma}$. $(\Sigma, \oplus, 0_{\Sigma})$ is a commutative monoid.

Despite the name, a choice operation on a set of values may return an element which
is not in the given set: for instance, the choice operation from probabilities will be summation, and for lattice elements it will be the least upper bound.

Example 5 Turning now to Figure 2.1(b), again considering it as a probabilistic system: the total probability of reaching $s_f$ from $s_i$ should be equal to $\alpha + \beta + \gamma + \delta$; assuming, of course, that the probabilities of choosing one of these paths are independent. The identity for summation is 0; there may be any number of impossible paths from $s_i$ to $s_f$, but they make no difference to the reachability.

We are now in a position to define sequential composition of relations with multiplicities.

**Definition 5** Let $R \in \Sigma^S \times T, R' \in \Sigma^T \times U$. Then $(R \cdot R') \in \Sigma^S \times U$ is defined as follows for all $s \in S, u \in U$:

$$(R \cdot R')(s, u) = \bigoplus_{t \in T} (R(s, t) \otimes R'(t, u))$$

![Diagram](image)

Figure 2.2: A system with multiset transitions.

Example 6 The transition relation $R$ shown in Figure 2.2 is a multi-relation on the state set $Q = \{q_i, q_a, q_b, q_f\}$. We want to find the connectedness of $q_i$ and $q_f$ in $R^2$. From the definition of relational product, we know:

$$R^2(q_i, q_f) = \sum_{q \in Q} (R(q_i, q) \downarrow R(q, q_f))$$
We consider the value of $0_{\Sigma}$ assigned to a pair to represent the intuitive notion that they are completely unrelated; and the value of $1_{\Sigma}$ indicating that they are maximally related. In composition of two connections between members, we assert that composing any connection with a $0_{\Sigma}$-valued connection should yield a compound connection which is also $0_{\Sigma}$-valued. We dub this the zero sequencing axiom:

$$0_{\Sigma} \otimes x = x \otimes 0_{\Sigma} = 0_{\Sigma}$$

for all $x \in \Sigma$.

These requirements for the behaviour of relation multiplicities dictate a particular choice of algebraic structure. It turns out that semirings provide exactly the properties needed for relational composition to be well-defined.

## 2.6 Semirings

**Definition 6 [32]** If $(\Sigma, \otimes, 1_{\Sigma})$ is a monoid, $(\Sigma, \oplus, 0_{\Sigma})$ is a commutative monoid, $\otimes$ distributes over $\oplus$ on the left and right, and the zero sequencing axiom holds, then $K = (\Sigma, \otimes, 0_{\Sigma}, \oplus, 1_{\Sigma})$ is called a semiring.

We have argued for necessity of all the semiring axioms except for distributivity of $\otimes$ over $\oplus$: we shall address it now. Consider Figure 2.3: in (a), a transition with value $a$ into an intermediate state precedes a choice between two possible transitions, valued $b$ and $c$ respectively. Either the first will be chosen, for a total path valuation of $a \otimes b$, or
the second will be chosen, for a total path valuation of $a \otimes c$. In effect this is exactly the same as the choice of transitions in (b). The distributivity axioms formalizes our intuition that composing a transition with a choice should be the same as factoring the first transition into each of the choices.

![Diagrams of semiring distributivity axiom](image)

**Figure 2.3:** Illustration of the semiring distributivity axiom.

There is a standard embedding $\eta$ of the set $\mathbb{B}$ of truth values into any semiring: it takes $\top$ to $1_\Sigma$ and $\bot$ to $0_\Sigma$. This embedding is a formalization of the intuitive notion that $0_\Sigma$ is like "false", and $1_\Sigma$ like "true".

The basic definition of a semiring is customarily extended in a few ways. Some of these have to do with order, which we shall use to formalize the ordering of paths by desirability:

**Definition 7** If there is a partial order $\sqsubseteq$ defined on $\Sigma$, then $K$ is called a partially-ordered semiring. If

$$a \oplus b = a \sqcup b$$

and

$$a \otimes b \sqsubseteq a \oplus b$$

then $K$ is a lattice-ordered semiring. If, for all $a, b \in \Sigma$, $a \sqsubseteq b$ if and only if $\exists c \cdot a \oplus c = b$, then $\sqsubseteq$ is called the natural order or difference order.

A semiring operation $\ominus$ is called a negation if:

$$\ominus 0_\Sigma = 1_\Sigma \quad (0\text{-}1 \text{ Negation I})$$

$$\ominus 1_\Sigma = 0_\Sigma \quad (0\text{-}1 \text{ Negation II})$$
If $\Theta$ is involute ($\Theta \circ a = a$), then it is called a complementation.

If $\Theta$ is idempotent (for all $\sigma, \sigma \circ \sigma = \sigma$), then $K$ is called an idempotent semiring (or additively-idempotent.¹)

**Definition 8** The identity relation on $S$ relative to the semiring $K$ is the relation $I \in \Sigma^{S \times S}$ such that for any $R' \in \Sigma^{S \times T}, R'' \in \Sigma^{T \times S}$:

$$(I \cdot R' = R') \land (R'' \cdot I = R'')$$

It can be shown that $I$ is the relation where:

$$I(s, t) = \begin{cases} 
1_\Sigma & \text{if } s = t \\
0_\Sigma & \text{otherwise}
\end{cases}$$

We call the support of a set with multiplicities in $K$ those elements of the universe which are not assigned the value $0_\Sigma$. In the event that the valuation is partial, the support consists of those elements which are assigned some value, but not $0_\Sigma$.

Finally, we define forward image and backward image of a relation: these notions formalize the way in which a $K$-valued relation transforms valuations of elements.

**Definition 9** The forward image of a valuation $v \in \Sigma^S$ under a relation $R \in \Sigma^{S \times T}$ is the valuation $FW(R, v) \in \Sigma^T$ defined as follows:

$$FW = \lambda R \in \Sigma^{S \times T}. \lambda v \in \Sigma^S. \lambda t \in T. \bigoplus_{s \in S} (v(s) \otimes R(s, t))$$

and backward image is similarly defined, taking a valuation in $\Sigma^T$ and yielding a valuation in $\Sigma^S$:

$$BW = \lambda R \in \Sigma^{S \times T}. \lambda v \in \Sigma^T. \lambda s \in S. \bigoplus_{t \in T} (R(s, t) \otimes v(t))$$

¹This terminology, as ever, is not universal. Some authors [4] use the term quasiring for what we call a semiring, and call a semiring what we have defined as an idempotent or additively-idempotent semiring. We have simply gone with what seemed to be the majority usage [26, 32, 8] and many other sources.
For fuzzy and multiple-valued sets and relations, image under a relation has been called shadow under a relation [48, 25], to emphasize the intuitive notion of approximativeness and possibility that such valuations are meant to convey.

2.7 Additional Structure

2.7.1 Optimistic and Pessimistic Choice

We mention here the informal notions of "angelic" and "demonic" nondeterminism. An angel is an adversary which, given a nondeterministic choice, always chooses the best possible one for our goal; and a demon always chooses the worst.

These two adversaries are present in spirit (appropriately enough) whenever system properties are stated in a temporal logic. For instance, in the branching-time logic CTL [38], it is possible to specify properties such as $\text{AF} \varphi$ - "in all possible futures, $\varphi$ becomes true", or $\text{EF} \varphi$ - "in some possible futures, $\varphi$ becomes true". The first property, rephrased, states that "even if the worst choice is always made, $\varphi$ will still become true at some point"; and the second, that "$\varphi$ may become true, but only if the correct choices are made".

Thus, "sometimes" is sometimes "in the best of all possible worlds". With this in mind, we will extend the standard definition of semirings to include not a single choice operator, but both an optimistic and a pessimistic one; we shall denote the new, pessimistic, choice operator by $\oplus^d$. We will continue to consider $0_\Sigma$, the identity element of $\oplus$, to denote the absence of any transition: and this absence means that even the "demon" cannot choose it.

This is a little unusual, so we shall illustrate it with a pair of examples.

Example 7 If the transitions are probabilistic, we consider $+$ to be the optimistic choice operation, with identity $0_\Sigma = 0$. If there are two paths with nonzero probability from state
a to state b, the probability of reaching b from a is their sum. All of the other paths can be summed in as well, since they do not affect the value.

As a pessimistic choice, it makes sense to choose \( \downarrow \); of all the paths, we choose the least likely. However, we want this to be the least likely one that it is still possible to choose; otherwise we will always get zero. So pessimistic choice must always be over the nonzero paths (the support set); that is, for any non-zero \( n \), \( n \downarrow 0 = 0 \downarrow n = n \), and \( 0 \downarrow 0 = 0 \). It has the identity \( id_d = 1 \); which in this case, as with many others (including lattices), is equal to \( 1_\Sigma \).

**Example 8** With transition multiplicities representing costs, the system expends a specific quantity of some resource every time a transition is taken. For the sake of discussion, we will think of this resource as consisting of pennies. The “angel” wants the system to spend as few pennies as possible to reach a desirable state; by contrast, the “demon” wants it to spend as many pennies as possible before reaching a desirable state - and if it is possible for the system to keep spending pennies without reaching that desirable state, then the demon will make the choices that result in that infinitely expensive outcome. If this is possible, then the pessimistic choice among path valuations should yield \( \infty \) as an outcome. If, however, there is a finite maximum to the number of pennies that can be spent before reaching the desirable state, then the proposition “in the worst case, eventually \( \varphi \)” should have that maximum as a value. The proposition “in the best case, eventually \( \varphi \)” will yield the minimum cost; this will only be \( \infty \) if that good condition is completely unreachable.

### 2.7.2 Implication

When we come to perform formal calculation over transition systems with multiplicities, we will sometimes need to deal with statements like “if there is a transition to some new state, then property \( \varphi \) must hold in that state”. We want a certain relationship to hold between pessimistic choice and implication: if there is a \( 0_\Sigma \)-valued (impossible)
transition, then the value of the implication should be identity for pessimistic choice.

\[
(0_\Sigma \rightarrow x) = \text{id}_d \quad \text{(Vacuity)}
\]

\[
(1_\Sigma \rightarrow x) = x \quad \text{(Necessity)}
\]

We require, as well, that if there is no transition (it has the value \(0_\Sigma\)), then the implication should never have the value \(0_\Sigma\), since it is always vacuously non-false. That is, that identity for optimistic choice should differ from identity for pessimistic choice, which seems only reasonable.

**Example 9** For quasi-boolean lattices, the quasi-complement \(\neg\) is a natural choice of negation operator. As well, we choose \(a \rightarrow b \equiv \neg a \cup b\) as the implication operator. For pessimistic choice, we will use \(\cap\). These choices reduce to the standard definitions when the lattice chosen is \(\mathbb{B}\).

**Example 10** When we deal with costs on transitions, we find again that generalizing logical operations is a little unusual. As the negation of any finite cost, we choose infinity; and as the negation of infinite cost, zero cost.

### 2.7.3 Transition Algebras

We put all of this additional structure into a single definition, of what we shall call a transition algebra. Such algebras are simply partially-ordered semirings with a negation, and definitions of pessimistic choice and implication added.

**Definition 10** A transition algebra \(K\) is a tuple \((\Sigma, \oplus, 0_\Sigma, \oplus^d, id_d, \ominus, 1_\Sigma, \subseteq, \Theta, \rightarrow)\) such that:

- \((\Sigma, \oplus, 0_\Sigma, \ominus, 1_\Sigma, \subseteq)\) is a partially-ordered semiring;
- \((\Sigma, \oplus^d, id_d)\) is a commutative monoid;
• $\Theta$ is a semiring negation;

• $id_d \neq 0_\Sigma$;

• For all $x \in \Sigma$, $(1_\Sigma \to x) = x$ and $(0_\Sigma \to x) = id_d$.

It should be emphasized, though, that in many definitions and results to follow the whole structure of a transition algebra will not be needed, only the semiring structure. We shall be thinking of a transition algebra as a partially-ordered semiring with some additional operations; and where the ordering, negation, and implication of a transition algebra are not relevant, we shall simply refer to semirings, with the tacit understanding that the additional structure can always be put back in.

In this chapter, we have developed the structure needed to define the transition relation of a system. In the next chapter, we shall use this structure to define transition systems with multiplicities, where the states are valuations on a set of variables, and thus generalize the familiar notion of Kripke structures [42].
Chapter 3

Finite-State Models with Multicplicities

3.1 Background

In this section, we present a formal definition of a finite set of worlds (states), connected by an accessibility relation with multiplicities. These multiplicities represent some sort of information about changes of state - that they have a level of probability or possibility, or a fixed cost, or an action from some alphabet, assigned to them.

Following this definition, we list some semirings that have been used in modelling, and extend them to transition algebras. Some constructions for producing more exotic transition algebras are introduced briefly, and we also present one example of "transition engineering" to deal with reset transitions.

3.2 Kripke Structures with Transition Multicplcities

A finite-state system model is generally constructed in one of two ways. The first is as a high-level description, during requirements analysis or early design: such models are usually specified using a state-based modelling language such as SCR [29] or Statecharts
The purpose of formal analysis of such a system is to check that a high-level design meets its specifications so that refinement may continue, and sometimes also to produce executable specifications for purposes of documentation and negotiation. Finite state models may also be produced from a complete program, by abstraction. Analysis of such models is called static analysis or abstract interpretation [18]. Static analysis is used to verify that already-written programs meet a specification, and also to infer properties of the program that will allow a compiler to optimize it further.

**Definition 11** A transition system with multiplicities in $K$, or a $K$-Kripke structure is a tuple $(S, S_0, K, R, I, A, V)$

- $K = (\Sigma, \oplus, 0, \oplus^d, id, \otimes, 1, \subseteq, \ominus, \rightarrow)$ is a transition algebra;
- $S$ is a set of states:
- $S_0 \subseteq \Sigma^S$ is an initiality function on states: informally, it determines "how initial" a state is.
- $R \subseteq \Sigma^S \times S$ is a transition relation with multiplicities in $K$;
- $V$ is a finite set of values for state variables; it is either $\Sigma$ or $\mathbb{B}$.
- $A$ is a set of variable symbols (atomic propositions);
- $I \subseteq V^S \times A$ is a valuation function - given a state $s$ and variable symbol $a$, yields the value of $a$ in state $s$.

In the classical case, such a structure is subject to the additional constraint that at least one transition out of each state must exist: there are no "sinks". We assume that this "Kripke condition" is met when each state has some outgoing nonzero transition; a stronger condition might be the requirement that the sum of all outgoing transitions from each state be $1_\Sigma$. If $K$ is the probabilistic semiring, this makes the transition relation stochastic [41]. For some applications, however, this may be too strong a condition.
Now we define the notion of a path in the structure, and its valuation.

**Definition 12** A path is an infinite sequence of states in $S$. The set of all paths is denoted by $S^\omega$: the set of all paths beginning with a specified state $s$ is denoted by $sS^\omega$; and a subsequence of a path $\pi$ is written $\pi(m:n) = \pi_m \pi_{m+1} \ldots \pi_n$. The expression $\pi(m: \infty)$ is the (infinite) suffix of $\pi$ starting at position $m$.

A path $\pi = s_0s_1s_2 \ldots$ has a value in the structure if:

$$\lim_{n \to \infty} \left( \bigotimes_{i \in [n]} R(s_i, s_{i+1}) \right)$$

exists; the value of the path is that limit.

In general, we restrict the paths considered to be those which have a defined value, and whose value is not $0_\Sigma$. We recall that is the *support* of the path valuation: the support of all paths is written $\text{Supp}(S^\omega)$, and the support of all paths beginning in state $s$ is $\text{Supp}(sS^\omega)$.

In passing, we note that this path valuation is the first valuation function we have encountered which is *partial*.

### 3.3 Multiplicities for Modelling

#### 3.3.1 Examples of Transition Algebras

Here we state some useful semirings for transition multiplicities, and extend them to transition algebras, also sketching the types of specifications associated with them.

- The *boolean semiring* $(\{T, \bot\}, \lor, \bot, \land, T)$ is used in ordinary finite-state systems where only the presence or absence of a transition is specified; this is sufficient where correctness relative to some boolean property is desired. As pessimistic choice we take $\land$, with identity $T$; and ordinary implication $\Rightarrow$. 
• Any distributive lattice is a semiring \((L, \sqcup, \sqcap, \downarrow, \top)\). This includes the boolean semiring and the fuzzy semiring \(F = ([0, 1], \uparrow, 0, \downarrow, 1)\). If the lattice is also quasi-boolean (as both the boolean and fuzzy lattices are), the quasi-complement serves as a negation, \(\sqcap\) as pessimistic choice, and generalized material implication as implication.

• The min-plus or Dijkstra semiring \(\mathbb{N}_\mathbb{L} = (\mathbb{N} \cup \{\infty, -\infty\}, \downarrow, \infty, +, 0)\), used to compute shortest paths: the interpretation in terms of costs make \(\uparrow\) a natural pessimistic choice operator, with identity \(-\infty\). For negation, we choose \(\ominus n = \infty\) for any finite \(n\), and \(\ominus \infty = 1\); and for implication, \(n \rightarrow y = n + y\) for finite \(n\), and \(\infty \rightarrow y = -\infty\). We dub this the transition algebra of costs. Note that a path is higher in the desirability ordering if its cost is lower; so the ordering of values is the reverse of the usual ordering on numbers.

• The algebra of probabilities \(P = (\mathbb{I}, +, 0, \times, 1)\); transition systems with multiplicities in \(P\) are also known as Markov decision processes [41]. The obvious negation is \(\ominus x = (1 - x)\), and \(\downarrow\) fits as a pessimistic choice operator. We leave the question of implication open; there are many different notions of probabilistic implication, which can be used to fit different applications – see, for instance, [16].

• There may be a finite alphabet of actions associated with transitions. For this case, the semiring \((2^\text{Act}^*, \cup, \emptyset, \cdot, \lambda)\) where \(-\) is catenation, and \(\lambda\) the empty word, is standard. Although this more general algebra is used for path valuations, transitions themselves only have multiplicities from the set \text{Act}. Intersection of path labels seems an obvious pessimistic choice, and implication, as with costs, is somewhat trivial: \(x \rightarrow y\) is just \(x \cdot y\) unless \(x = \{\}\), in which case it is the identity for pessimistic choice, \(\text{Act}^*\). An application of this algebra is in the computation of synchronizing sequences, sets of inputs which always return a system to a certain "good" set of states.
A comprehensive overview of semiring theory, covering a large number of different instances of the structure and some of their prior uses in modelling can be found in the reference of Golan [26].

Both the probabilistic and fuzzy semirings make use of the interval $I$ as a carrier set; only the operations differ. In practice, the use of the fuzzy semiring reflects what has been called possibilistic rather than probabilistic reasoning; the value attached to the transition defines a level of confidence that it exists to be taken at all, rather than how frequently it will be taken in a sequence of independent trials.

### 3.3.2 Construction of Transition Algebras

More elaborate transition algebras can be built from simpler ones by some standard construction, such as direct products and power, quotients under equivalence relations, power algebras, and disjoint unions. We give some applications where these constructions are of use.

**Example 11** In parallel composition of transition systems with costs, we may wish to model the fact that different system components may be using distinct resources; thus, transitions in the composition may have multiplicities in some power of $\mathbb{N}$, say $\mathbb{N}_i^2$. The transition multiplicity $(x, y)$ signifies that the change of global state requires $x$ units of the resource used by component 1, and $y$ of that used by component 2.

**Example 12** In order to model incomplete knowledge about costs, we can label transitions with intervals: for instance, "this transition may take anywhere between 3 to 5 time units". The interval algebra is a quotient of the power set $2^\mathbb{N}$; the congruence $\theta_i$ is the one which relates all sets whose maximum and minimum points are the same:

$$A \equiv_{\theta_i} B \iff \inf A = \inf B \land \sup A = \sup B$$

The set of intervals of any partially-ordered set $\Sigma$ - the quotient $2^\Sigma/\theta_1$ - is denoted $\text{Int}(\Sigma)$. 


3.3.3 An Example of Transition Engineering

Some timed automaton models [2, 33] have designated "reset transitions", which pull the value of a system clock back down to zero. Although the continuous-time nature of these models is beyond the scope of this thesis, we still find this a valuable idea that would add to the expressiveness of transition systems with costs.

So, can we add this refinement to a transition algebra, rather than specifying an extra map on transitions? It turns out, if we consider the cost transition algebra, that we can add a special "reset" symbol to the carrier set.

This yields the new carrier set \( \mathbb{N} \cup \{\infty, -\infty, \preceq\} \) where \( \preceq \) is the special reset value. Our first intuition is that a path on which we first take some finite time, and then a reset, the value should be equal to zero: and if we take first a reset transition, and then some finite time, that finite time should be the final value. Resetting any number of times should be equivalent to one reset, so we make it idempotent. With this in mind we propose the following properties of \( \preceq \), for all \( n \in \mathbb{N} \):

\[
\begin{align*}
    n + \preceq & = 0 \\
    \preceq + n & = n \\
    \preceq + \preceq & = \preceq
\end{align*}
\]

Of course, we wish the algebra to remain a semiring when only the optimistic choice and sequencing operations are chosen: Firstly, the identities must remain identities:

\[
\begin{align*}
    0 + \preceq & = \preceq + 0 = 0 \\
    \infty \downarrow \preceq & = \preceq \downarrow \infty = \preceq
\end{align*}
\]

Secondly, the operations \( \downarrow \) and \( \uparrow \) must continue to commute, and \( \infty \) must still be an annihilator for \( + \): if, for instance, an infinite delay is followed by a reset, then the reset
can never occur to set it back to zero, so \( \infty + \leq = \infty \). At this point, we take the decision based on intuition that a reset takes zero time:

\[
\begin{align*}
n \uparrow \leq &= \leq \uparrow n = n \\
n \downarrow \leq &= \leq \uparrow n = \leq \\
\infty + \leq &= \leq + \infty = \infty
\end{align*}
\]

Since reset takes zero time, we also decide that \( \varnothing \leq = \infty \), and that \( \leq \rightarrow x = x \). Distributivity must be preserved as well:

\[
\begin{align*}
x + (y \downarrow z) &= x + y \downarrow x + z \\
(y \downarrow z) + x &= y + x \downarrow y + z
\end{align*}
\]

We show in Figure 3.1 operation tables which yield the desired properties. In these tables, \( n \) stands for any non-zero, non-infinite element of \( \mathbb{N} \).

As far as minimum and maximum are concerned, a reset should behave exactly like zero, as it does here.

Note that this highlights a few subtle points about transition algebras, since \( \text{id}_d \) is not in the domain of any operation other than \( \varnothing^d \), and not in the range of any operation other than \( \rightarrow \). When this is the case, in order to perform computations over the system safely, it is required that \( \text{id}_d \) never be used to label a transition; it only arises when we perform pessimistic choice over implications.

This example is an attempt to illustrate one of the driving ideas behind the current work: that all of the customization of labelled transition systems can be pushed inside the transition algebra. Instead of considering a "transition system with reset transitions", we consider an ordinary transition system, whose transition algebra includes resets as a special value.
Figure 3.1: Operation tables for the cost algebra with resets. $n$ represents any non-zero, non-infinite cost.

(a)\[
\begin{array}{c|cccc}
+ & n & 0 & \infty & \triangle \\
\hline
n & n & n & \infty & 0 \\
0 & n & 0 & \infty & 0 \\
\infty & \infty & \infty & \infty & \infty \\
\triangle & n & 0 & \infty & \triangle \\
\end{array}
\]

(b)\[
\begin{array}{c|cccc}
\downarrow & n & 0 & \infty & \triangle \\
\hline
n & n & 0 & n & \triangle \\
0 & 0 & 0 & 0 & 0 \\
\infty & n & 0 & \infty & \triangle \\
\triangle & \triangle & 0 & \triangle & \triangle \\
\end{array}
\]

(c)\[
\begin{array}{c|cccc}
\uparrow & n & 0 & -\infty & \triangle \\
\hline
n & n & n & n & n \\
0 & n & 0 & 0 & 0 \\
-\infty & n & 0 & -\infty & \triangle \\
\triangle & n & 0 & \triangle & \triangle \\
\end{array}
\]

(d)\[
\begin{array}{c|cccc}
\rightarrow & n & 0 & \infty & \triangle \\
\hline
n & n & n & \infty & 0 \\
0 & n & 0 & \infty & 0 \\
\infty & -\infty & -\infty & -\infty & -\infty \\
\triangle & n & 0 & \infty & \triangle \\
\end{array}
\]
Chapter 4

Propositional Temporal Calculi

4.1 Background

In this chapter we examine how the addition of transition multiplicities to a finite-state system makes possible the generalization from temporal logic to a temporal calculus for expressing computations about the system. There is no formal reason for this terminological shift - however, in practice, the term "logic" seems to be applicable when the underlying set of values is a lattice, or a set of probabilities, but a little remote from intuition when it is some other algebra.

We shall show how there are several choices involved in making this generalization, yielding three proposed varieties of branching-time temporal calculus. These provide a way of expressing computations about such properties of a system as the maximal probability of reaching a failure state, the minimum cost of attaining a goal, or the certainty of reaching a desirable state in one step.

4.2 Temporal Logic

Temporal logic extends standard propositional (or first-order) logic to deal with modalities of time - propositions may be said to become true at some point in some possible
future, or in all possible futures, or to hold at all times (starting from the implied present time). A temporal logic formula can be modelled by a sequence of assignments to the propositional variables; this corresponds in a natural way to a path in a finite state machine. This correspondence has been called “possible-worlds semantics” [42]. In any given state – a “world” – some purely propositional formula can be modelled by the assignment in that state. If, in some state, every assignment reachable in a single step models the formula, we say that the state models the formula in the next step. If, regardless of the steps taken from the state, some formula is satisfied at some point in the future (“some”, in this context, meaning “at least one”), we say that the state models the formula eventually. Other temporal properties can also be expressed.

Now, since we have described accessibility relations with multiplicities from some set of values, and algebraic rules for composing them, we propose to extend temporal logic so that statements about the cost, or probability, or certainty of reaching some future state can be made and used for automated reasoning.

4.3 From Logic to Calculus

4.3.1 State, Transition, and Path Values

We assume that it is already decided that transition values are taken from the carrier set $\Sigma$ of the transition algebra. Values of propositional variables in a state may either be from $\Sigma$, or they may be boolean. The situation where they are boolean is a little more intuitive, so we shall assume it.

Ordinary temporal logic assigns boolean values to both paths in the Kripke structure, and to individual states. For instance, the CTL* path formula $F \varphi$ – “along this path, eventually $\varphi$” – assigns the value $\top$ to those paths in the structure along which, at some point, $\varphi$ becomes true; and $\bot$ to all the others. Applying path quantification to a path formula yields a state formula, which assigns values to states: the state formula $EF \varphi$
gives the value $T$ to a state if at least one of the paths starting out from that state is given the value $T$ by $F \varphi$.

Given that we have chosen to give transitions values from $\Sigma$, it seems reasonable that path valuations should also be from $\Sigma$. If we take some path formula like $Fp$, where $p$ is a boolean-valued state variable, it seems that the most reasonable option is that this formula gives a value to a path which indicates something about the manner in which it reaches a state where $p$ holds. If it never does, then its value ought to be $0_\Sigma$. It may give the probability of the path reaching $p$, or the certainty, or the cost: whatever intuitive interpretation we put on the values of $\Sigma$ from the transition algebra.

However, the question of what sort of values we wish to put on states remains open. In probabilistic model-checking [27, 34], questions such as “from the starting state, the system reaches a good state with probability greater than or equal to 0.98” are sometimes posed. This is a boolean state formula, which implicitly performs a computation over the probabilistic path formulae starting in that state.

All of the temporal calculi presented here will provide $\Sigma$-valued path formulae. The temporal calculus $KCTL_1$ will provide boolean state formulae, and $KCTL_2$ and $KCTL_3$ will provide $\Sigma$-valued state formulae. In point of fact, $KCTL_3$ only provides state formulae.

### 4.3.2 Path Quantification

In branching-time temporal logics, quantification over paths in the Kripke structure is performed: one can make statements such as “in all next states, $\varphi$ holds” and “in at least one next state, $\varphi$ holds”.

It seems clear that “in at least one next state” generalizes to optimistic choice over path valuations, and that these path valuations are determined by the product (in the transition algebra) of the value of $\varphi$ in the second state of the path with the value of the transition between the first and second states. In the boolean case, this reduces
back down to $\exists \pi. R(\pi_0, \pi_1) \land [\varphi]_{\pi_1}$, which is precisely what we should expect: and in the cost algebra, it becomes $\inf_{\pi} R(\pi_0, \pi_1) + [\varphi]_{\pi_1}$: which can be read intuitively as "the minimum (over all paths) of the sums of the cost of getting to the next state with the cost of making $\varphi$ true in the destination state".

However, there seem to be two possible choices for how to generalize "in all next states" or "along all paths". One is in terms of demonic nondeterminism: we really mean "in the worst case". The other is in terms of implication: if a transition is possible at all, then something should hold on the other side of the transition. For impossible transitions or paths, we do not care what happens.

**Example 13** In the cost algebra, if a transition is impossible, this is indicated by the fact that it has infinite cost. It seems that this is a situation where what we really want is the worst among all non-infinity-valued, that is, possible, paths. Thus, to evaluate an expression such as $AX p$, we should perform pessimistic choice (maximum) over all of the paths out of the current state that are not infinite-valued; although, of course, if $p$ does not hold in the second state of such a path, the path formula $X p$ will still give the value $\infty$ to it. This path valuation by formula - how the path satisfies the property - must be distinguished from the path valuation in the system, which is merely the product of the individual transitions taken.

**Example 14** On the other hand, consider a multiple-valued system. We feel that the semantics of "on all paths, next $\varphi$" should be that "the greater the certainty of a path, the higher the likelihood of $\varphi$ being true in the second step should be". This correlation is not obtained simply by taking the minimum of the transition value with the value of $\varphi$ in the next state; instead, we want to use implication: there are several multiple-valued or fuzzy implication operators in the literature [48], but we choose generalized material implication, $x \rightarrow y \equiv \neg x \uplus y$. This yields the "for all" quantification defined in the multiple-valued temporal logic $\forall$CTL [15].
4.4 Three Temporal Calculi

4.4.1 Purposes

Each of the three variants described here will produce formal results regardless of the choice of transition algebra; however, those results may make more or less sense based on the user's intuition about the meaning of transition values. Different forms of path quantification (choice over paths) seem to suit different transition algebras – for possibilistic transitions, such as fuzzy or lattice-valued systems, implication or some form of correlation between the possibility of a path and the value of a formula along it, seems to be essential, whereas with costs, or actions, where the concern is more with the quality of the path rather than its presence or absence, “for all” path quantification seems to be the coarser-grained notion of pessimistic choice among values of paths in the support.

4.4.2 KCTL$: Valued Paths, Boolean States

This temporal calculus is related, in approach, to the probabilistic branching-time logic PCTL [27]. The path formulae are valued in the set of multiplicities, but the state formulae are boolean, obtained by some threshold over valuations of the state's outgoing paths. This allows for expressions with intuitive meaning such as “with probability $> 0.75$, eventually $\varphi$ holds”. or “with certainty $\geq 0.5$, $\varphi$ holds until $\psi$ becomes true”. A probability models an level of unpredictability in the world: an event is observed to happen with a certain probability distribution. A possibility value assigned to an event, on the other hand, is representative of an uncertainty in our knowledge of a system. Given two states, if there is a transition between them with probability 0.5, that transition will be taken, in the limit, half the time that the system is in the first state; however, if the transition has possibility 0.5, the modeller has only that level of confidence that the transition exists to be taken at all. A survey of possibilistic reasoning is found in [36].

This choice of calculus requires that the set $V$ of propositional variable valuations be
### Figure 4.1: Grammar for $L_{KCTL}$.  

$\text{Literal}$ is any literal of the transition algebra; $\text{Atom}$ any propositional variable symbol.

In this calculus, only optimistic choice of the transition algebra is used: so actually, only the semiring structure is strictly necessary to apply it. The syntax appears in Figure 4.1, and the semantics in Figure 4.2. State formulae can either be standard propositional logic expressions, or a path-threshold expression $[g] \bowtie v$ where $g$ is a path formula, $\bowtie$ some comparison operator. For the sake of expressibility in ASCII we have used standard comparison operators $\leq, <, \geq$ etc., but of course it is important to recall that $K$ is not required to be totally ordered, but only partially ordered. The path-threshold operation can be read as a predicate on the sum of all path valuations.

A state valuation is converted into a path valuation using the standard embedding $\eta$ of $\mathbb{B}$ into a semiring; the state formula is evaluated at the first state along the path, and

<table>
<thead>
<tr>
<th>$\text{Syntax}$</th>
<th>$\text{Semantics}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Formula} \rightarrow \text{StateFormula}</td>
<td>\text{PathFormula}$</td>
</tr>
<tr>
<td>$\text{StateFormula} \rightarrow \text{StateFormula} \lor \text{StateFormula}</td>
<td>\neg \text{StateFormula}</td>
</tr>
<tr>
<td>$\text{PathFormula} \rightarrow \text{PathFormula} \lor \text{PathFormula}</td>
<td>\neg \text{PathFormula}</td>
</tr>
<tr>
<td>$\text{PathFormula} \rightarrow \text{PathFormula} \lor \text{PathFormula}</td>
<td>\neg \text{PathFormula}</td>
</tr>
<tr>
<td>$\text{CompareOp} \rightarrow \leq</td>
<td>\geq</td>
</tr>
</tbody>
</table>
then converted into either $0_\Sigma$ or $1_\Sigma$. This embedding has different intuitions in different algebras: for a state formula $\varphi$, it could be read as “the certainty of $\varphi$ holding”, “the probability that $\varphi$ holds”, “the cost to make $\varphi$ true”, or “the input required to make $\varphi$ true”.

This intuition is extended to the path formulae as well: for instance, we can read $F \varphi$ as “the maximum certainty with which $\varphi$ holds along the path”, or “the probability that this path reaches a state where $\varphi$ holds”, or “the minimal cost of reaching a state where $\varphi$ holds along this path”.

$$
\begin{align*}
\llbracket p \rrbracket_s &= I(s, p) \\
\llbracket \neg f \rrbracket_s &= \neg \llbracket f \rrbracket_s \\
\llbracket f \land f' \rrbracket_s &= \llbracket f \rrbracket_s \land \llbracket f' \rrbracket_s \\
\llbracket f \lor f' \rrbracket_s &= \llbracket f \rrbracket_s \lor \llbracket f' \rrbracket_s \\
\llbracket \mu g \nu \rrbracket_s &= (\bigoplus_{\pi \in \pi(s)} \llbracket g \rrbracket_\pi) \triangleright v
\end{align*}
$$

$$
\begin{align*}
\llbracket f \rrbracket_\pi &= \eta(\llbracket f \rrbracket_{\pi_0}) \\
\llbracket X g \rrbracket_\pi &= \llbracket g \rrbracket_\pi(1 : \infty) \otimes R(\pi_0, \pi_1) \\
\llbracket F g \rrbracket_\pi &= \bigoplus_{k \geq 0} \llbracket g \rrbracket_\pi(k : \infty) \\
\llbracket G g \rrbracket_\pi &= \bigotimes_{k \geq 0} \llbracket g \rrbracket_\pi(k : \infty) \\
\llbracket g \cup g' \rrbracket_\pi &= \bigoplus_{k \geq 0} (\llbracket g' \rrbracket_\pi(k : \infty)) \otimes (\bigotimes_{0 \leq j < k} \llbracket g \rrbracket_\pi(j : \infty)) \\
\llbracket g \bowtie g' \rrbracket_\pi &= \bigotimes_{k \geq 0} (\llbracket g' \rrbracket_\pi(k : \infty)) \otimes (\bigotimes_{0 \leq j < k} \llbracket g \rrbracket_\pi(j : \infty))
\end{align*}
$$

Figure 4.2: Semantics for $L_{KCTL^1}$, where $f, f'$ are state formulae and $g, g'$ are path formulae, and the model is implicit. $\triangleright$ represents any one of the comparison operators $<, \leq, >, \geq, =, \neq$.

**Example 15** Properties with thresholds arise naturally in many situations. For instance, $[idle \cup request] \geq 0.9$ expresses the property “with probability greater than or equal to 0.9, the system stays idle until there is a request”; $[F stable] \leq 50$. the property “the system reaches a stable state in less than 50 time units” (or expending less than 50 power units, depending on the resource being used).
Figure 4.3: Grammar for $L_{KCTL^*_2}$.

4.4.3 KCTL^*_2: Valued Paths and States, Optimism and Pessimism

KCTL^*_2 assigns values from the set of transition multiplicities to states as well. The path quantifiers A and E are generalized to optimistic and pessimistic choice over the valuations on paths out of a state; and this valuation is limited to the support of paths, that is, those which are considered possible in the Kripke structure.

The syntax for KCTL^*_2 appears in Figure 4.3: a subset of this language, where path formulae may only appear quantified, forms the syntax for KCTL_3 as well. The semantics of KCTL^*_2 appear in Figure 4.4.

Example 16 A partial road-map of central Asia appears in Figure 4.5. The transition algebra of costs is used. The question “how long is the shortest path from Samarkand to Kyzyl?” is answered by $[EF\text{ Kyzyl}]\text{Samarkand}$. Here we use KCTL^*_2, and consider “being in a certain city” as a boolean propositional variable. Note that, using the standard embedding of $\mathbb{B}$ into the cost algebra, $\eta(\top) = 0$, which makes intuitive sense: if we are
Figure 4.4: Semantics for $L_{KCTL^2}$, where $f, f'$ are state formulae and $g, g'$ are path formulae, and the model is implicit.

\[
\begin{align*}
[f]_s = \eta(I(s, p)) & \quad [X g]_\pi = [g]_\pi(1 : \infty) \otimes R(\pi_0, \pi_1) \\
[-f]_s = \ominus([f]_s) & \quad [F g]_\pi = \bigoplus_{k \geq 0} [g]_\pi(k : \infty) \\
[f \wedge f']_s = [f]_s \otimes [f']_s & \quad [G g]_\pi = \bigotimes_{k \geq 0} [g]_\pi(k : \infty) \\
[f \vee f']_s = [f]_s \oplus [f']_s & \quad [g \cup g']_\pi = \bigoplus_{k \geq 0} ([g']_\pi(k : \infty) \otimes \bigotimes_{0 \leq j < k} [g]_\pi(j : \infty)) \\
[E g]_s = \bigoplus_{\pi \in Supp(s_S^w)} [g]_\pi & \quad [g \mathbin{R} g']_\pi = \bigotimes_{k \geq 0} ([g']_\pi(k : \infty) \otimes \bigotimes_{0 \leq j < k} [g]_\pi(j : \infty))
\end{align*}
\]

in some place, the cost to reach it is certainly 0.

The all-pairs shortest path problem was expressed as a symbolic model-checking problem in [5] as well: we believe that the explicit specification of the associated algebraic structure makes for a clearer presentation.

Example 17 Consider again the road-map of Figure 4.5, and the universal property $AX$ Tashkent, which we evaluate again in state Samarkand. Only those paths out of Samarkand which are possible are considered: those beginning with either (Samarkand) (Tashkent), or (Samarkand)/(Bokhara). Of those, $X$ Tashkent has value $150 + 0 = 150$ in the first set, and $200 + \infty = \infty$ in the second. The pessimistic choice between them yields $\infty$.

By contrast, if we checked the property $AX$ (Tashkent $\lor$ Bokhara), then we should expect to get the maximum of 150 and 200, namely 200. This is the maximum distance which it is possible to go before satisfying the formula (Tashkent $\lor$ Bokhara).
4.4.4  **KCTL₃: Valued Paths and States, Implication**

Although the only significant difference between KCTL₃ and KCTL₂ is in the definition of the path-quantifier $A$, this difference requires a different approach in the definition of the calculus' semantics. We define the existential next-state operator $\text{EX}$ using optimistic choice over products, and the universal next-state operator $\text{AX}$ using pessimistic choice over *implications*: and all other temporal operators are then defined in terms of these two.

Since no path formulae appear unquantified, this is a variant of plain CTL, rather than of CTL*. 

**Example 18** In Figure 4.7 we see an example of a lattice-valued Kripke structure. Figure 4.7(a) is the four-valued boolean lattice $4\text{Bool}$. Consider the value of $[\text{AX } x]q₀$ in

Figure 4.5: A somewhat fanciful road-map of central Asia.
\[ p_s = \eta(I(s, p)) \]
\[ \neg f_s = \phi([f]s) \]
\[ [f \land f']_s = [f]s \land [f']_s \]
\[ [f \lor f']_s = [f]s \lor [f']_s \]
\[ [\text{EX } f]_s = \bigoplus_{\pi \in S_{\text{Supp}(s')}} R(s, \pi_1) \otimes [g]_{\pi_1} \]
\[ [\text{AX } f]_s = \bigoplus_{\pi \in S'_{\text{Supp}(s')}} (R(s, \pi_1) \otimes [g]_{\pi_1}) \]
\[ [\text{EF } f]_s = [f]s \lor [\text{EX } \text{EF } f]_s \]
\[ [\text{AF } f]_s = [f]s \land [\text{AX } \text{AF } f]_s \]

Figure 4.6: Semantics for \( L_{\text{KCTL}_3} \), where \( f, f' \) are state formulae, and the model is implicit.

The model of Figure 4.7(b):

\[ [\text{AX } x]_{q_0} \]
\[ = \bigoplus_{\pi \in S} (R(q_0, \pi_1) \rightarrow [x]_{\pi_1}) \]
\[ = (R(q_0, q_1) \rightarrow [x]_{q_1}) \lor (R(q_0, q_2) \rightarrow [x]_{q_2}) \lor (R(q_0, q_0) \rightarrow [x]_{q_2}) \]
\[ = FT \rightarrow \eta(T) \lor (TF \rightarrow \eta(T)) \lor (TF \rightarrow FF) \rightarrow \eta(F) \]
\[ = (FT \rightarrow TT) \lor (TF \rightarrow TT) \lor (FF \rightarrow FF) \]
\[ = (TF \lor TT) \lor (TF \lor TT) \lor (TT \lor FF) = TT \]

This result makes sense: for each transition (even including the FF-valued one from \( q_0 \) to \( q_0 \)), the value of \( x \) in the successor state is at least as true as the transition.

4.4.5 Valued State Variables

In \( \chi\text{CTL} \), the multiple-valued temporal logic used in the \( \chi\text{bel} \) framework for reasoning over inconsistent specifications [15, 13, 14], we used propositional variables from the
Figure 4.7: (a) The lattice for the four-valued logic $\mathbf{4Bool}$, and (b) a model with multiplicities from $\mathbf{4Bool}$.

transition logic.

If state formulae are boolean-valued, as in $\mathbf{KCTL}_1^*$, there are two possibilities: either a standard mapping from $\Sigma$ into $\mathcal{B}$ can be defined (which will throw away some information), or the syntax can be extended slightly so that any comparison on a state variable can be used as a propositional variable:

$$[p \bowtie v]_s = I(s, p) \bowtie v$$

In $\mathbf{KCTL}_2^*$ and $\mathbf{KCTL}_3$, things are much simpler. It is simply necessary to make the semantics of a propositional variable $p$ into its value $I(s, p)$ in a state, removing the $\eta$ which embeds boolean truth values into the transition algebra.

4.4.6 Multiple-Valued Model-Checking

$\mathbf{KCTL}_3$, when $\mathcal{K}$ is a finite quasi-boolean lattice, has been described as the multiple-valued temporal logic $\mathbf{\chiCTL}$ [15, 13]. This logic is quite tractable; all paths have valua-
tions, and given a formula and a model, the value of the formula in the model can always be decided.

The application driving this particular choice of transition multiplicity is dealing with inconsistency in specifications. Several stakeholders provide incomplete descriptions of a system, which reflect their particular domain of concern - such descriptions are called viewpoints [20]. These are merged using a formal procedure. and this merged model retains information about disagreements between stakeholders: for instance. stakeholder A may have permitted a transition between two system states that stakeholder B did not. The value attached to the transition represents this disagreement: it may be completely information-preserving, such as “A says yes. B says no. the rest don't care” or the more abstracted “there was a disagreement”.

The value of a \(\mu\)CTL formula in such a merge of viewpoints is a measure of the degree to which these disagreements affect the truth-value of the property specified by the formula. If it has the value \(T\) - the top value of the lattice - then we are able to conclude that any disagreements which may exist are not relevant to the property at all. and resolving them may be deferred until a later stage in development. If it is below \(T\), then the disagreements do have an effect: and the precise value can be used to pinpoint the sources of this disagreement, particularly if we have used an information-preserving merge which yields results like “A disagrees with B about this formula”.

### 4.5 Evaluation of Formulae

Given a model. and a formula in one of these logics, we want to compute the value of the formula in the model. This is the model-checking problem. The simplest approach - though not necessarily the best - is the explicit-state approach: the model is represented as a list of pairs \((q, q')\) of states, together with the value of the transition between them. and the semantics of a formula is represented as a mapping from states to values.
For small cases this is not difficult. However, as the number of states becomes large, the complexity of manipulating these tables of values grows beyond the bounds of feasibility. The problem is that the explicit-state representation does not make any use of structure or symmetry in the model; even if most of the states are not reachable, they still need to be in the table. It would be better to work with a representation of the state-valuation (and transition-valuation) function which can leverage such structure for a more compact representation than a simple enumeration of values.

Such representations are normally called *symbolic* [38], although this is a slightly misleading term. The standard symbolic representation of transition relations of classical Kripke structures, and of the semantics of temporal logic formulae over such models, is the *binary decision diagram* [11]. In the following chapter, we shall present the basic ideas underlying this kind of representation of functions over a finite domain, and discuss extensions suitable for representing the $K'$-Kripke structures of this thesis.
Chapter 5

Decision Diagrams

5.1 Definitions

In this section we discuss decision diagrams. Binary decision diagrams (BDDs) are the data structure of choice for representing transition relations in symbolic model-checking [38]. Multi-terminal binary decision diagrams (MTBDDs) [23], also called algebraic decision diagrams (ADDs) [4], have been proposed and implemented for representing the transition values of probabilistic systems, with boolean state variables. In some recent work [14], we have used decision diagrams whose terminal nodes and internal variables are both drawn from certain finite lattices; to represent those, we have used MDDs [46] and shared BDDs [39].

We begin with a description of the ideas behind decision-diagram representations, and then present the formalism. Decision diagrams have also been called "branching programs" [1], and as such can be considered as much a programming language, with a simple denotational semantics, as a mere data structure. This programming language has the convenient property that transformations on functions can be expressed in simple syntactic terms.
5.1.1 Decomposing Finite Functions

We consider a variable domain \( D = \{d_0, d_1, \ldots, d_{m-1}\} \) of finite size \( m \) and a range \( R = \{r_0, r_1, \ldots, r_{n-1}\} \) of size \( n \) (which may be \( \infty \)). Since, for the definition of decision diagrams, it is only the numbering that is significant and not the actual values, so for the remainder of this section we will use \([m]\) in place of \(D\) and \([n]\) in place of \(R\): the actual sets can be easily recovered at any stage.

Decision diagrams are used to denote functions of the type \([m]^k \to n\), functions from length-\(k\) vector over a domain of size \(m\) into a size-\(n\) range.

The finite domain allows the hierarchical decomposition of a function in \(k\) variables into \(m\) functions of \((k-1)\) variables; these can, in turn, themselves be recursively decomposed, until 0-variable functions – constants – are reached. This process yields a decision tree representation of the function, and by the removal of constant nodes and identification of identical nodes, the decision tree can be transformed into a reduced decision diagram: this reduction, in general, requires less space, and also permits constant-time checking for equality of functions. This canonicity property – the property that identical functions are represented by identical graph-nodes – is dependent on every function being decomposed in the same order.

**Definition 13** Let \( f : [m]^k \to [n] = \lambda x_0. \cdots. \lambda x_k. \varphi(x_0, \ldots, x_k) \). From basic \(\lambda\)-calculus we know that \(\lambda\)-abstraction commutes, and we can choose any \(x_i\) and bring it to the front. so:

\[
f = \lambda x_i. \lambda x_0. \cdots. \lambda x_{i-1}. \lambda x_{i+1}. \cdots \lambda x_k. \varphi(\cdots)
\]

For any \(i \in [k], j \in [m]\), the \(j\)-th cofactor of \(f\) with respect to \(x_i\) is:

\[
f[x_i/j]
\]

Intuitively, this is the function in \(k-1\) variables obtained by assigning the value \(j\) to the variable \(x_i\), and allowing all the other variables to remain free; where \(x_i\) is implicit, we write it just as \(f_j\).
A $k$-ary function can be reconstructed from its cofactors using a case-statement, which takes in the list of $(k - 1)$-ary cofactors and a single number $i \in [m]$ as arguments, and returns the $i$th function. The list of cofactors is expressed as a mapping $[m] \rightarrow f$.

$$\text{case}_{k} \equiv \lambda j : [m].\lambda c : [m] \rightarrow ([m]^{k-1} \rightarrow [n]).c(j)$$

The subscripted $k$ is used because the function has a slightly different signature for each $k$: in general, however, we shall simply use the unsubscripted case and assume that the appropriate $k$ will be selected.

**Proposition 1** A function whose variables have finite domain can be expressed as a case-statement of its cofactors:

$$f = \lambda j.\text{cas}(j, \lambda j.f[x_{i}/j])$$

In other words, evaluation at a decision-tree node (using case) is the inverse of decomposition.

This is known as the Boole-Shannon expansion [11] or cond-normal-form [37]. We represent this expansion graphically as shown in Figure 5.1: the children of the node labelled with $x_{i}$ are the cofactors. It is easy to see that they, too, can be decomposed recursively, until constant values are reached: this yields a decision tree.

If all of the cofactors with respect to a variable $x_{i}$ of a function are identical, it is said to be constant in $x_{i}$. The set of all variables for which a function is non-constant is known as its support.

The decision tree produced by this recursive expansion has two sources of redundancy: the first is common cofactors, where the same subtree appears in two or more different places. Once these are identified and merged, it is possible to discover and remove redundant comparisons - these are constant nodes, all of whose children are identical. In this fashion we obtain a more compact structure which is no longer a tree but a DAG: a decision diagram.
Figure 5.1: The graphic representation of the Boole-Shannon expansion of function $f$ at $x_i$, for $m = 4$.

### 5.1.2 Syntax

Syntactically a reduced and ordered decision diagram is a graph with several constraints on its edges.

**Definition 14** A reduced, ordered $(m, n)$-decision diagram in $k$ variables is a tuple $U = (U_0, N, T, m, n, k, Lab, Val, Child)$ where:

- $N$ is a set of nonterminal nodes:

- $T$ is a set of terminal nodes, with $m = |T|$;

- $U_0$ is the root node:

- $Lab : N \to [k]$ labels a nonterminal node with the number of one of the $k$ variables (this numbering gives a total ordering on the variables);

- $Val : T \to [m]$ labels a terminal node with the number of one of the $m$ possible values:

- $Child : N \times [n] \to (N \cup T)$: $Child(u, i)$ yields the $i$th child of node $u$.

- $Image : (N \cup T) \to 2^{|m|}$, for any node, gives the values of the terminal nodes reachable from it.
The following axioms describe the restrictions on a valid decision diagram, for all nodes $u, u' \in N \cup T$:

\[
\begin{align*}
\forall i \cdot \text{Child}(u, i) \in T \lor \text{Lab(Child}(u, i)) > \text{Lab}(U) & \quad \text{Orderedness} \\
(u \in N) \Rightarrow \exists j \cdot \text{Child}(u, i) \neq \text{Child}(u, j) & \quad \text{Reducedness 1} \\
(\text{Lab}(u) = \text{Lab}(u') \land (\forall i \cdot \text{Child}(u, i) = \text{Child}(u', i))) \Rightarrow u = u' & \quad \text{Reducedness 2} \\
\end{align*}
\]

\[
\begin{align*}
\forall u \in T \Rightarrow \text{Image}(u) = \{\text{Val}(u)\} & \quad \text{Image 1} \\
\forall u \in N \Rightarrow \text{Image}(u) = \bigcup_{i \in [m]} \text{Image(Child}(u, i)) & \quad \text{Image 2} \\
\end{align*}
\]

By a slight abuse of notation, we will use $\text{Child}(u)$ to refer both to the node $u' \in N \cup T$ which it returns, and to the DD with $U_0 = u$.

### 5.1.3 Semantics

An $(m, n)$ decision diagram $U$, in $k$ variables, denotes a function $[U] : [m]^k \rightarrow [n]$. This denotational semantics can be simply expressed in the following manner, which dictates computation by tracing a single path from root to leaf:

\[
[U] = \lambda v_1, \ldots, v_{k-1}. \text{ if } (U_0 \in T) \text{ then } \text{Val}(U_0) \\
\quad \quad \quad \quad \quad \quad \text{else case}(v_{\text{Label}}(U_0).[\text{Child}(U, 0)](v_0, \ldots, v_{k-1}), \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{\ldots}, \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad [\text{Child}(U, k - 1)](v_0, \ldots, v_{k-1}))
\]

Intuitively, the value of the variable labelling the root node $U_0$ (that is, $v_{\text{Label}}(U_0)$) is used to choose which of the child nodes to evaluate recursively: in this recursive evaluation, that child becomes the root node of a new, smaller DD which is evaluated in turn. The process bottoms out when a subdiagram with $U_0 \in T$ is evaluated.

The absence of cycles in the decision diagram guarantees that function evaluation terminates in no more than $k$ steps.

It has been shown that the syntactic properties of reducedness and orderedness allow the semantic property of canonicity of a decision-diagram representation of a function to be easily proven.
Theorem 1 [46] Reduced and ordered \((n, m)\)-DDs are canonical: that is, if there are two nodes \(U, U' \in (N \cup T)\) such that \([U] = [U']\), then \(U = U'\).

5.1.4 Representing State Sets and Transition Relations

Assume we have some \(K\)-Kripke structure with \(|V| = k\). We take \(V'\) as all the primed copies of \(V\), and take \(V \cup V'\) as the set of variables of our decision diagram. As defined, the transition relation is a \(2k\)-ary function in the variables \(V \cup V'\). then

In order to do reachability analysis, we need to be able to transpose state sets: that is, take a state set which is solely a function of \(V\), and prime all of the variable labels so that it represents a successor state. Then, to complete forward or backward image computation, we need to abstract out the primed variables by summation — this, as we shall show later, turns out to be equivalent to matrix multiplication in the semiring.

5.2 Decision Diagram API and Implementation

It is useful to consider decision diagrams as an implementation of an abstract class `FunctionObject`, which represents functions over finite domains. We list the essential methods of this class and then for decision diagrams, we state what additional methods must be implemented. Several enumerated types are used: `var` is the type of variable symbols, `value` the type of domain values, `rvalue` the type of range values, and `operator` the type of operators on the range.

- `FunctionObject RESTRICT(var v, value x)`: return the new `FunctionObject` resulting from restricting variable \(v\) to have value \(x\).
- `bool IS_CONSTANT()`: true if the function is constant, false if it has some dependency on its variables
- `rvalue GET_VALUE()`: return the value of a constant function.
- FunctionObject GET CONSTANT(rvalue val): construct a constant FunctionObject representing some element of the range.

- FunctionObject APPLY(operator o): apply a unary operator to the FunctionObject. returning the resulting FunctionObject.

- FunctionObject APPLY(operator o, FunctionObject op2): apply a binary operator to the referenced function object and a second operand, and return the resulting FunctionObject.

- list of (list of (var, value)) ALL PREIMAGE(rvalue x): return all of the sets of variable assignments that yield x as an output.

Having stated these as the basic methods which need to be implemented by a decision diagram library. there is really only one which is specific to it:

- DD BUILD(FunctionObject f): build a decision diagram from a FunctionObject.

- DD SIMPLIFY(FunctionObject f): minimize the decision diagram with respect to a domain of interest specified by f: every variable assignment outside the support of f is considered to be a "don't care" condition which never arises in the inputs.

The principal benefit of introducing this level of abstraction is to underline that any representation for functions over finite domains is equally expressive, and BUILD can be used to cast one to the other at need. The only differences between the basic decision diagrams of this section, and the varieties which will be introduced in the next, are in terms of efficiency.

The standard method of implementation is the use of a unique table, which depends on the canonicity property: a node in the decision diagram is uniquely identified by its Label and Child functions, and so they can be used as a key in a dictionary of nodes. Each node is then represented by its index in the unique table, and an inverse table - the dictionary - keyed by its label and the indices of the children - is used to guarantee that no redundant
nodes are ever inserted into the unique table. The function MAKEUNIQUE either returns a node with a specified variable label and children, or creates a new one. Implementation details of decision diagrams, concentrating on (2, 2)-DDs but easily generalizable to the (m, n)-case, can be found in the tutorial of Andersen [3] of the survey of Somenzi [45]. MTBDDs [23] are (2, \infty)-DDs: although, of course, with a finite range the full infinite set of terminal nodes is never actually constructed.

5.3 Decision Diagram Varieties

We present here two varieties of decision diagram that show potential for the compact representation of transition relations with multiplicities in arbitrary semirings.

- *Shared BDDs* [39], which we used in [14] to represent systems with transitions from finite distributive lattices: these, or some variation, seem eminently suited for use with finite semirings, where the state variables are either boolean, or drawn from the same set as the transitions.

- *Edge-valued DDs* [47], where sharing of subgraphs is possible whenever they represent different functions that can be transformed into each other by some invertible operation; in the boolean case, this operation is either negation or the identity.

This section gives the definitions, properties, and some examples of these two varieties: algorithms for creating and manipulating them appear in the Appendix.

5.3.1 Shared Decision Diagrams

The key idea behind shared decision diagrams is to decompose elements of a range into sets. Generally this decomposition is some form of binary encoding; a range of 8 values can be represented in 3 bits, splitting any single function onto that range into 3 binary-valued functions. The motivation is that this will increase space efficiency by promoting
sharing of common subexpressions, and this, in practice, turns out to be justified [39].

The principal application of shared decision diagrams to date has been in verifying arithmetic circuits, which have an obvious binary decomposition. We extend this notion slightly, and discuss shared decision diagrams for finite semirings, presenting a technique which serves, in some cases, to produce a good binary decomposition.

**Definition 15** [39] A shared DD $U$, for an function of type $[m]^k \rightarrow n$, is an $(m, 2)$-decision diagram having, instead of a single root node $U_0$, a collection $U_0 = \{ U^1_0, U^2_0, \ldots, U^n_0 \}$ of root nodes. It is associated with a decoding $\Delta : \mathbb{B}^r \rightarrow [n]$, such that:

$$ [f]_r = \lambda v_0 \ldots \lambda v_{k-1} : \Delta([U^1_0](v_0, \ldots, v_{k-1}), \ldots, [U^n_0](v_0, \ldots, v_{k-1})) $$

Here $[\cdot]_r$ indicates that shared DD semantics, as distinct from ordinary, single-rooted DD semantics, are being defined.

**Example 19** A circuit outputs numbers between 0 and 7. Since we can represent these numbers using 3 bits, we can represent this circuit as a shared DD with 3 root nodes $\{ U_0, U_1, U_2 \}$. The decoding $\Delta : \mathbb{B}^3 \rightarrow [8]$ is defined as follows:

$$ \Delta(x, y, z) = 4x + 2y + z $$

giving the usual place-value binary encoding.

It should be clear that it is always possible, given an encoding which is the inverse of $\Delta$, to construct a shared DD (which is really a set of $(m, 2)$-DDs).

**Theorem 2** Let $U'$ be an $(m, n)$-ary decision diagram in $k$ variables representing a function $f^u : [m]^k \rightarrow [n]$, with $n$ finite. Then for any $2 \leq n' \leq n$ and $\Delta : [n']^r \rightarrow [n]$, it is possible to construct a shared decision diagram $U'$ with $r$ root nodes such that $f^{u'} = f^{U'}$.

**Proof 2** Let $\text{ENC} = \Delta^{-1} : [n] \rightarrow [n']^r$ be the encoder for $[n]$.

For all $j \in [r]$, we define $\text{ENC}_j : [n] \rightarrow [n']$ as the $j$th component of the encoding.
As shown previously, given a function \( f : [m]^k \rightarrow [n] \) we can compose it with \( \text{ENC}_j \) to yield \( \text{ENC}_j \circ f : [m]^k \rightarrow [n'] \). Then we can perform a Boole-Shannon expansion (with a set variable ordering) produce a canonical \((m, n')\)-DD representing \( \text{ENC}_j \circ f \) for each \( j \). Since the individual DDs are canonical, the whole set is as well; they remain so when we merge identical nodes to produce a single, multiply-rooted shared DD.

In the next subsection we will investigate in more detail the construction of a “good” \( \Delta \) where \( n' = 2 \).

The important use of these results is when \( n' = 2 \); it allows us to represent any \( n \)-ary characteristic function as a shared BDD. Although this implementation choice may result in a loss of efficiency in operations on decision diagrams, it facilitates reuse of existing BDD technology, which is quite mature \cite{45}; and also promotes the merging of common subexpressions.

For \((m, \infty)\)-DDs, of course, this procedure cannot be used unless the range is abstracted in some way to make it finite. Two feasible ways to do this are with integer arithmetic with a set maximum value, or, if the range is floating-point, with a limit on precision. Such an abstraction makes verification approximate, but then this is frequently needed to ensure convergence in any case.

### 5.3.2 Binary Representations

**Definition 16** \cite{26} An additive set of generators for a semiring \( K \) is a subset \( A \subseteq \Sigma - \{0\Sigma \} \), such that for any \( s \in \Sigma \), there is a canonical subset \( A' \subseteq A \) having the property that:

\[
\begin{align*}
  s &= \bigoplus_{a \in A'} a \\
  a \in A' &\quad \text{for which} \quad (a \cup b = \ell) \Rightarrow (a = \ell \lor b = \ell)
\end{align*}
\]

**Example 20** If \( K \) is a finite distributive lattice, the additive set of generators is exactly the set of join-irreducible elements: those \( \ell \) for which

\[
(a \cup b = \ell) \Rightarrow (a = \ell \lor b = \ell)
\]
We note that non-distributive lattices are generally not quite semirings: they lack \( \otimes \) over \( \oplus \) distributivity.

**Theorem 3** [26] Given an idempotent semiring \( K \) with an additive set of generators \( G \), and a function \( \text{ENC} : \Sigma \rightarrow 2^G \) which takes any \( x \in \Sigma \) to its decomposition:

\[
\text{ENC}(x \oplus y) = \text{ENC}(x) \cup \text{ENC}(y)
\]

and

\[
\text{ENC}(x \otimes y) = \text{ENC}(x) \cap \text{ENC}(y)
\]

This allows us to express any function with \( K \) as its range as a shared binary-terminal DD with \(|G|\) root nodes, and APPLY the operations \( \oplus \) and \( \otimes \) by applying the standard boolean operations of \( \land \) and \( \lor \) pairwise on the root nodes.

### 5.3.3 Edge-Valued Decision Diagrams

Ordinary BDDs are commonly optimized with *complement edges*. These edges indicate that the node they terminate in is to be complemented: if it represents the function \( f \) by itself, then with the complement edge it represents \( \neg f \). This allows for the sharing of common subexpressions and their complements: so \( x \land y \) would be represented by the same subgraph as \( \neg x \lor \neg y \). In order to maintain the property of canonicity, it is required that the low edge always be non-complemented.

This idea has been generalized [35, 47] to DDs representing functions with a numeric range; in edge-valued BDDs (EVDDs), an edge transformer adds a specified quantity to its terminus, whereas in factored edge-valued BDDs (FEVBDDs), the edge transformer multiplies by a specified quantity \( a \) and then adds a quantity \( m \). This is the group of affine transformations.

All of these proposed sets of edge valuations have the common property of being groups on the range of the diagram. In Figure 5.2, we show why: the node shown in the
Figure 5.2: Normalization of edge transformers using inverses.

The figure is being created using MAKEUNIQUE: in order to maintain uniqueness, it needs to be normalized, with the transformer on the lowest edge being the identity. On the left, the transformer on the lowest edge is $f$; so $f^{-1}$ is calculated, and applied (on the left) to all of the transformers on the outgoing edges. This normalized node is either already in the unique table, or it is created; MAKEUNIQUE then returns that table entry, together with the edge transformer $f$ leading into it.

**Definition 17** A normalized edge-valued DD is an $(m,n)$-DD with these additional elements:

- $ET$, a group of $[n] \rightarrow [n]$ functions, the edge transformers;

- $EV al : N \times (N \cup T) \rightarrow ET$, a function assigning transformers from $ET$ to edges. We define the auxiliary function $ChildEdge : N \times [m] \rightarrow ET$ as $ChildEdge(u,k) = EV al(u, Child(u,k))$.

- $RootEdge$, the transformer assigned to the root node $U_0$.

We no longer consider that a function is represented solely by a node and its descendants, but instead by a pair of a node (and its descendants) with an incoming edge transformer.

For normality, it is necessary that, for all nonterminal nodes $u \in N$,

$$EV al(u, Child(u, 0)) = id$$
and also that there be precisely one terminal node:

\[ |T| = 1 \]

The definition of reducedness also changes:

\[ u \in N \Rightarrow \exists i, j \cdot ((ChildEdge(u, i) \cdot \text{Child}(u, i)) \neq ((ChildEdge(u, j) \cdot \text{Child}(u, j))) \]

Reducedness 1

\[ (\text{Lab}(u') = \text{Lab}(u) \land (\forall i \cdot \text{Child}(u, i) = \text{Child}(u', i))) \Rightarrow u = u' \]

Reducedness 2

The condition "Reducedness 1" stipulates that at least two outgoing edges differ in either their labels or their destinations, and possibly both; however, it is possible that each edge leads to the same destination, as long as at least two of the labels differ.

Finally, the definition of Image requires different handling; it has the domain \( ET \times (N \cup T) \), the set of edge-transformer and node pairs.

\[ u \in T \Rightarrow \text{Image}(e, u) = e(\text{Val}(u)) \]

\[ u \in N \Rightarrow \text{Image}(e, u) = \bigcup_{i \in [m]} e(\text{Image}(\text{ChildEdge}(u, i), \text{Child}(u, i))) \]

Again we abuse notation somewhat: we use \( \text{Child}(u, k) \) to refer to the node which is the \( k \)th child of node \( u \), and also to the normalized edge-valued DD with \( \text{Child}(u, k) \) as its root node and \( \text{ChildEdge}(u, k) \) as its root edge.

With this in mind, the semantics are defined in a manner almost identical to ordinary DDs:

\[ \llbracket U \rrbracket = \lambda v_1, \ldots, v_{k-1}. \begin{cases} \text{RootEdge}(\text{Val}(U_0)) & \text{if } (U_0 \in T) \text{ then} \\ \text{RootEdge}(\text{case}(v_{\text{Label}(U_0)}, [\text{Child}(U, 0)](v_0, \ldots, v_{k-1})), \ldots, \\ [\text{Child}(U, k - 1)](v_0, \ldots, v_{k-1})) & \text{else} \end{cases} \]

The transformations chosen must all be one-to-one and onto in order to be invertible. In the boolean case, \( \neg \) and \( \text{id} \) are the only two invertible functions on \( \mathbb{B} \), so they are the
only candidates. As we shall see, the situation is a little more complicated even for finite semirings: and idempotence of a binary operation makes it non-invertible in general. since it ceases to be a one-to-one operation.

**Theorem 4** Normalized edge-valued \((m, n)\)-DDs for finite \(n\) with the group of \(n\) cyclic shifts as edge transformers are canonical.

**Proof 4** The proof, as with all DD canonicity proofs, proceeds by induction on the number of variables that the represented function depends on.

**Base case.** There is exactly one terminal node \(t\), and for any possible constant value \(v \in \{n\}\), there is exactly one \(Sh_i\) such that \(v = Sh_i(\text{Value}(t))\).

**Induction step.** Assume that all functions of less than \(k\) variables have a canonical representation. Let the function \(f : [m]^k \rightarrow [n]\) be a function in \(k\) variables. Performing the Boole-Shannon expansion:

\[
f = \text{case}(x_1, f_0, f_1, \ldots, f_{m-1})
\]

and by the induction hypothesis, there are unique nodes \(u_0, \ldots, u_{m-1}\) with incoming edge valuations \(Sh_{j_0}, \ldots, Sh_{j_{m-1}}\) such that

\[
\forall i \cdot f_i = Sh_{j_i}([u_i])
\]

If all of the \(u_i\) are the same, and all the \(j_i\) are the same, then the function is constant in \(x_1\), and \((u_i, Sh_{j_i})\) is the normalized edge-valued DD for \(f\). The node \(u\) with incoming edge \(Sh_{j_0}\), children \(u_0, \ldots, u_{m-1}\) and child edge transformers \(Sh_0, Sh_{j_1-j_0}, \ldots, Sh_{j_{m-1}-j_0}\) is the canonical representation for \(f\): any other node representing \(f\) would have the same children and child edges, and thus, by reducedness, it would be the same node.

**5.3.4 Example**

We consider a finite distributive lattice as the range. Unfortunately, both meet and join (with the second operand chosen) are idempotent, and thus neither is invertible! We must look elsewhere for a suitable group of edge transformations.
For this example, we shall re-use the lattice-valued model given in Chapter 4. In Figure 4.7(a), we saw the distributive lattice for the four-valued logic $\mathbf{4Bool}$. Negation is defined as pointwise: that is, $\neg \text{FT} = \text{TF}, \neg \text{FF} = \text{TT}$.

In this example, we use two-valued state variables, for the sake of simplifying the graphical presentation: in general, though, the state variables also come from this logic. The Kripke structure is shown in Figure 4.7(b).

We first represent this as an ordinary $(2, 4)$-DD, as in Figure 5.3. We notice that this is not much of a reduction from a decision tree: redundant tests have been eliminated, but there are no common subexpressions. Either one of the two alternative representations might do better.

![Figure 5.3: A $(2, 4)$-DD for the system from Figure 4.7(b).](image)

First, we consider representing this as a shared $(2, 2)$-DD. In order to do this, we use the two-bit binary representation of elements from $\mathbf{4Bool}$, using the atoms TF and FT.
This diagram is shown in Figure 5.4. We have not really improved, but we have not created very many more nodes, either.

![Diagram of a shared BDD with two root nodes](image)

Figure 5.4: A shared BDD, with two root nodes, for the system from Figure 4.7(b).

Of course, to perform one meet or join on the function represented this way requires doing two ANDs or ORs on the shared diagrams, which can result in a loss of efficiency.

As an alternative, we consider keeping the multiple-valued range, but using a group of edge transformers (larger than \( \{id, \neg\} \)) to help share subgraphs. Since we can’t, in general, undo joins or meets in a manner analogous to addition or subtraction – if we could, the lattice would be a *semifield*, not just a semiring – we must think a little more about what these edge transformers should be.

The approach of Sack et al. [43] uses not edge valuations, but another class of nodes
that perform a cyclic shift on the values returned by their children. This approach sacrifices canonicity; we argue that by considering these "shift nodes" as edge transformers, and enforcing normality - the low edge must be the identity - we can obtain a representation which may be slightly less compact, but will be canonical.

We propose, as an arbitrary mapping of the lattice elements into \([4]\), the lexicographic order \(FF < FT < TF < TT\), and consider the group \(G\) of cyclic shifts; this order induces these shifts are addition modulo 4. For instance, \(S_{h_2}(FF) = TF; S_{h_1}(TT) = FF\). The identity is \(S_{h_0}\), and \(S_{h_1} \cdot S_{h_j} = S_{h_{i+j \mod 4}}\); thus \((S_{h_i})^{-1} = S_{h_{-1 \mod 4}}\).

Using such shifts, the diagram of Figure 5.5 also represents, canonically, the multiple-valued transition relation of Figure 4.7(b). Unfortunately, this small example does not contain any additional subexpressions which edge transformers allow to be shared.

The diagram has exactly one terminal node, since the value of a function at a variable assignment is determined by composition of the edge transformers. Consider the transition from \(s_0\) to \(s_2\): \(x\) is false, \(x'\) true, \(y\) false and \(y'\) false. Tracing the edge transformers along the path - including the one going into the root - we obtain \(+0 \cdot +0 \cdot +2 \cdot +0 \cdot +0\). yielding \(+2\). which applied to the single terminal node gives a value of 2, corresponding to the logic value \(TF\).

In this example, we have shown two possible alternatives to standard DDs for representing a transition relation with multiplicities. Using shared DDs allows for reuse of existing BDD technology, and promotes the sharing of subexpressions, at some cost in the complexity of applying operations. Using edge-valued DDs holds the possibility of promoting greater sharing of subexpressions without performing a binary decomposition of the function, but is not so standard a technology. The choice, just from looking from this small example, is far from clear: comparisons on larger benchmarks are called for.

Decision diagrams allow for compact representation of transition relations and state-valuations in \(K\)-Kripke structures. In the next chapter, we will describe how they are manipulated to compute such state-valuations. These manipulations are, in general,
Figure 5.5: An edge-valued DD for the system from Figure 4.7(b). $S_{h,i}$ is represented as the annotation '+'$i$'.

considerably more efficient than those required for explicit-state model-checking. We will give symbolic model-checking algorithms for a fragment of $\mathcal{KCTL}_3$: these algorithms use the methods provided by a decision diagram library, and the underlying representation may either by ordinary $(m,n)$-DDs, or one of the two varieties described in this section.
Chapter 6

Symbolic Reachability Analysis with Multiplicities

6.1 Reachability Analysis

The essential component of symbolic model-checking is reachability analysis. In a graph with two-valued edges, a state is either reachable from another, or it is not; when there are transition multiplicities (edge weights in the relation graph), we have degrees of reachability, which we wish to compute.

For the general case of semiring-valued transition relations, it makes sense to think of the relation as a matrix.

Definition 18 Let $K = (\Sigma, \oplus, 0_\Sigma, \ominus, 1_\Sigma)$ be a semiring, $R \in \Sigma^{S \times T}$ be a $K$-valued relation, and $\alpha : S \to |S|, \beta : T \to |T|$ be numberings of $S$ and $T$. Then the matrix of $R$, $M^R$, is defined as follows. For all $s \in S, t \in T$:

$$M^R_{\alpha(s), \beta(t)} = R(s, t)$$

Under this definition, it can readily be shown that sequential composition of relations is equivalent to multiplication of their matrices.
For $K$-Kripke structures both $S$ and $T$ are the states: we can consider them as value assignments to $V$ and $V'$ respectively. $K$-valued sets can be thought of, then, as vectors; row vectors are pre-states, and column vectors are post-states.

Two matrices with the same dimensions can be added, entry-wise, in the usual manner: and multiplication by a scalar as well. However, since $\otimes$ is not required to be commutative, $rM \neq Mr$, in the general case, for a matrix $M$ and scalar $r$.

The square matrices $M_{n,n}(K)$ over a semiring $K$ themselves form a semiring, with matrix addition as $\oplus$, and matrix multiplication as $\otimes$.

In addition to the usual operations, the Kronecker product of matrices is useful in defining asynchronous parallel composition of systems [34].

**Definition 19** Let $A$ be an $n \times n$ matrix over $K$, and $B$ be an $m \times m$ matrix. Then the Kronecker product $A \otimes_k B$ is an $nm \times nm$ matrix such that:

$$(A \otimes_k B)_{i_1n+j_1,n_2+j_2} = a_{i_1,j_1} \otimes b_{i_2,j_2}$$

for all $0 \leq i_1, j_1 < n$, $0 \leq i_2, j_2 < m$. The Kronecker sum is defined as follows:

$$A \oplus_k B = (A \otimes_k I_m) \oplus (I_n \otimes_k B)$$

where $I_k$ is the $k \times k$ identity matrix (with $1_\Sigma$ on the main diagonal and $0_\Sigma$ elsewhere).

Conventionally, the symbol $\otimes$ alone is used for Kronecker product, but we have already used that for semiring multiplication.

![Figure 6.1: Two simple state machines.](image)

**Example 21** The state machines $S_1, S_2$ in Figure 6.1 have the transition matrices

$$M_1 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
respectively. Their Kronecker product is as follows:

\[ M_1 \otimes_K M_2 = \begin{bmatrix} 0 \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & 1 \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \]

which is the transition matrix for the synchronous product of \( S_1 \) and \( S_2 \), as shown in Figure 6.2.

![Figure 6.2: Synchronous composition (via Kronecker product) of \( S_1, S_2 \) from Figure 6.1.](image)

The backward-image computations involved in the \( \textbf{EX} \) and \( \textbf{AX} \) properties can be expressed as matrix multiplication; in the first case with \((\oplus, \otimes)\) as addition and multiplication, and in the second case with \((\oplus^d, \rightarrow)\).

### 6.2 Convergence and Approximation

The limit operator of the algebraic \( \mu \)-calculus [5] is shown by the authors to converge to a fixed-point under certain conditions. We hope, in future work, to state such conditions for various classes of semirings: determining, for a given transition algebra, which constraints on systems make which temporal logic formulae decidable. This is of considerable importance.
In order to assure converge of the symbolic model-checking algorithm for the algebraic $\mu$-calculus, Baier and Clarke do not require absolute convergence, but merely convergence to within some specified tolerance, which can be tuned to ensure some approximate result. A similar, but not yet explored, approach would be to use interval analysis [30] instead; this makes the approximation explicit.

6.3 Symbolic Model-Checking $\text{KCTL}_3$

There are four inputs to the model-checking algorithm:

- The transition algebra $K$;
- The transition relation $R$ of the model;
- A $\text{KCTL}_3$ formula containing only $\text{EX}$, $\text{AX}$, $\text{EU}$, $\text{AU}$ and propositional formulae;
- A maximum number of iterations.

The symbolic algorithms are shown in Figures 6.3 and 6.4. We show only algorithms for $\text{EX}$, $\text{AX}$, $\text{EU}$, and $\text{AU}$; $\text{EF}$ and $\text{AF}$ are can be obtained using the identities $\text{EF } \varphi = E[1\Sigma \cup \varphi]$ and $\text{AF } \varphi = A[1\Sigma \cup \varphi]$.

Since we do not have, in general, the identity $\text{AX } \varphi = \Theta \text{EX } (\Theta \varphi)$, we must also implement $\text{AX}$, which is not needed in ordinary CTL.

Until operations are implemented here using a threshold mechanism: they bail out after a certain number of iterations, which must be set by the user. A better method would be to define a metric on $\Sigma$-valued functions and use it to define approximate convergence. In some cases – both $\text{EU}$ and $\text{AU}$ for finite lattices, $\text{EU}$ for the algebra of costs – the fixed-point computation can be shown to always converge.

The model-checking algorithms return a valuation on the state sets. In the classical case, this is reduced to a boolean value by intersecting the set of states returned by model-checking with the initial set of states of the model, and checking this intersection
Figure 6.3: \textbf{EX} and \textbf{AX} (optimistic and pessimistic backward image) algorithms for KCTL₃.

for emptiness. Here, we wish to use the implication operator of the transition algebra: "insofar as a state is initial, it should be in the result of model-checking". We formalize this as follows:

\[
MC(S₀, \varphi) \equiv \bigoplus_{s \in S} S₀(s) \to \llbracket \varphi \rrbracket_s
\]

In the boolean case, this specializes back to set containment checking; for fuzzy or multiple-valued model-checking, it produces an answer based on both the certainties of \textit{transitions} in the system, and the certainty that the system begins in some state. In the algebra of costs, $S₀(s)$ gives, for state $s$, the cost incurred simply by beginning execution in that state.
The interpretation of the result depends on the intuition underlying the choice of transition algebra. During the modelling phase, the designers have been using a mental mapping from informal system descriptions into formal ones; interpreting the results of analysis requires that this process be reversed, in a sense. In Section 4.4.6, we discussed both of these informal phases with regards to \(\chi\)CTL model-checking; with probabilities and costs, we have straightforward everyday intuitions about the meanings of values that can be applied. If more exotic algebras are used, then more thought is required: it seems, though, that the motive for introducing more complex multiplicities should throw some
light on the meaning of computations in the system. However, this needs to be explored further.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

As stated at the outset, this work is an attempt to place system modelling and verification with weighted transitions in a more general setting, parameterized by the algebra used for transitions.

In order to deal with path quantification in branching-time logic, we have had to add some extra structure to the semirings which have conventionally been used for transition multiplicities in automata theory. Different notions of path quantification have also dictated a slight proliferation in varieties of temporal calculus, based on how we wish to generalize it.

We have discussed standard multiple-valued decision diagrams (MDD/MTBDDs) as a representation option, and also shared and edge-valued DDs. Previous treatments of these structures have assumed either boolean or numeric range for functions; the more algebraic treatment used here will, we hope, make the reuse of this technology for more exotic ranges easier.
7.2 Future Work

The close link between model-checking and linear algebra has been noted before here. for instance by Fujita et al. [23], who proposed decision-diagram representation and manipulation of sparse matrices. Linear algebra also appears to be of use in the important general task of deciding, relative to a given transition algebra $K$, what conditions upon a system make which temporal-logic formulae decidable.

The engineering of transition algebras itself is also an interesting task. There are many exotic constructions which may turn out to be of practical use: the use of interval algebras, for instance, and also of mappings from the set of intervals over a set $S$ into a semiring $K$ with carrier set $\Sigma$. Such mappings are members of $2^{\Sigma^{Int(S)}}$, and are known as histograms. We hope to develop more tools for this task, and look at several applications in detail: most immediately, histograms show promise for probabilistic abstract interpretation [40].

The other engineering task of interest, as stated before, is the design of decision diagrams, or indeed of other symbolic representations, for different algebras. The major question in this area is whether it is better to create tailored decision-diagram varieties, or to find clever binary representations of the characteristic functions of transitions with multiplicities.

One limitation is the failure to deal with timed automata in the sense of Alur and Dill [2]. In this formalism, there is a finite set of "control points", but within each control point one or more real-valued clock variables may vary in continuous fashion, subject to an invariant for that state. There is some work [34] extending this formalism with probabilities: and also some recent work [7] extending it with costs on transitions.

7.3 Acknowledgements

This thesis in its current form is an extension of work done on the $\chi$cek multiple-valued model-checker; Marsha Chechik and Steve Easterbrook are the architects of this project.
It was also strongly influenced by discussions with E.C.R. Hehner on probabilistic semantics of concurrent programs with discrete time.

The thinking behind this work was helped along substantially by the University of Toronto Formal Methods Reading Group, particularly Albert Lai, Victor Petrovykh, and Steve Easterbrook. I would also like to think Alessandro Cimatti and Marco Pistore of ITC-IRST, Trento, Italy, for valuable insights into symbolic representations, and Ansgar Fehnker of the University of Nijmegen, the Netherlands, for an enlightening discussion of model-checking with costs.

Gratitude for moral support is also due to Michael McMaster, Marna Nightingale, April Whitehead, Kim Kofmel, Catherine Crockett, Colin Hinz, Gemma Files, Stephen Barringer, Melani Litwack, Sian Reid, Jennifer Faer, Joan Devereux, Joanna Devereux, Cecily Devereux, Jeremy Devereux, and Stephanie Wood. The total ordering imposed on this set by linear text is, it should be noted, completely arbitrary.
Appendix A

Decision Diagram Operations

A.1 Shared BDDs

The function APPLY applies a function on the range to two multiply-rooted DDs: it returns a list of roots. All functions are implemented in terms of ordinary \((m, 2)\)-DD operations: these operations are prefixed with \texttt{BITLEVEL} to distinguish them from the shared-DD implementations. The reader is referred to [3] for ordinary decision diagram algorithms. It is assumed that the encoding \texttt{ENC} and decoding \texttt{DEC} are available; the member \texttt{root} of a shared DD \(u\) is an array of the root nodes, and the constant \texttt{root.length} contains the number of root nodes in a shared DD.

```plaintext
function APPLY(u:DD, u':DD, op:operator) returns DD

bitLevelOp := bitLevelOpOf[op]; // map word-level op to bit-level

foreach (0 ≤ i < u.root.length)
    newRoot[i] := BITLEVELAPPLY(u.root[0], u'.root[i], bitLevelOp);
return newRoot;
```

76
function GetConstant(v: rvalue) returns DD
encoding := ENC(v)
foreach i < root.length
    newRoot[i] := BITLEVELGETCONSTANT(encoding[i])
return newRoot

function Restrict(u: DD, x:value) returns DD
bitLevelOp := bitLevelOpOf[op]; // map word-level op to bit-level
foreach 0 ≤ i < u.root.length
    newRoot[i] := BITLEVELRESTRICT(u.root[i], x)
return newRoot

The algorithm for ALLPREIMAGE for shared DDs presented here is certainly not an optimal one, but it illustrates conceptually what needs to be computed. For a specified range value, a variable assignment is in the preimage if and only if, for all the elements in the encoding of the value, the assignment is a satisfying one for the corresponding root-node of the shared BDD; and also for all elements not in the encoding of the value, the assignment is non-satisfying.

function ALLPREIMAGE(u:DD, x:rvalue) returns list of (list of (var, value))
encoding := ENC(v)
foreach i < u.root.length
    if (encoding[i] = 1)
        assignments[i] := BITLEVELALLSAT(root[i])
    else
        assignments[i] := BITLEVELALLSAT(BITLEVELAPPLY(¬, root[i]))
return (∩, assignments[i])
A.2 Normalized Edge-Valued DDs

The type edge offers three operations: \texttt{EDGECOMPOSE(edge1, edge2)} which returns the new edge resulting from the composition of the two arguments; \texttt{EDGEINVERT(edge)} which returns the inverse of the parameter; and \texttt{EDGEAPPLY(edge, value)} which applies the edge transformer to a value in the range.

```plaintext
function MAKEUNIQUE(label:var, children:array of DD, edges:array of edge)
  returns (edge, DD)

  outgoingEdge := EDGEINVERT(edges[0]);
  foreach i \(0 \leq i < \text{children.length}\)
    edges[i] := EDGECOMPOSE(outgoingEdge, edges[0]);
  if (label, children, edges) \(\in\) uniqueTable
    newNode := uniqueTable.lookup(label, children, edges);
  else
    newNode := uniqueTable.insert(label, children, edges);
  foreach i
    newNode.image := newNode.image \cup\ EDGEAPPLY(edges[i], child[i].image)
  return (outgoingEdge, newNode)
```

Restriction of a variable to a value is not very different from the non-edge-valued case. Consider what happens at a node labelled with \(x_i\), when we restrict variable \(x_i\) to value \(i\). All incoming edges to the node are “glued” directly to the \(i\)th outgoing edge, bypassing the node. When these edges have transformers attached to them, it is necessary only to take the added step of composing the transformers. This is shown in Figure A.1.
function RESTRICT(eu: edge. u:DD. v:var. x:value) returns (edge. DD)

if (Label(u) > var)
  return (eu, u)
else if (Label(u) < var)
  foreach i \in [m]
    (newEdge[i]. newChild[i]) :=
      RESTRICT(ChildEdge(u.i). Child(u.i), var. value)
  return MAKEUNIQUE(Label(u). newChild. newEdge)
else // Label(u) = var
  return (EDGECOMPOSE(eu, ChildEdge(u, value)). Child(u. value))

Figure A.1: Restriction in edge-valued DDs.
function APPLY( u:DD, eu:edge, v:DD, ev:edge, op:operator) returns (edge, DD)
if (ISCONSTANT(u) ∧ ISCONSTANT(v))
    return GETCONSTANT(EDGEAPPLY(eu, GETVALUE(u)) op
        EDGEAPPLY(ev, GETVALUE(v)))
else if (Label(u) = Label(v))
    foreach i ∈ [m]
        (newEdge[i]. newChild[i]) := APPLY(Child(u, i),
            EDGECOMPOSE(u, ChildEdge(i, u)),
            Child(v,i). EDGECOMPOSE(v, ChildEdge(i, v)). op))
    return MAKEUNIQUE(Label(u), newChild. newEdge)

function FINDALLPREIMAGE(u:DD, eu:edge, v:VALUE) returns
    list of (list of (var, value))
if u ∈ T // terminal node
   // if it is the correct node, return the empty assignment
   if (v = EDGEAPPLY(eu, Val(u))) return ()
   // otherwise raise an exception
   else return ⊥
else // nonterminal node
result := () // start with empty list
foreach i ∈ [m]
result := catenate (map (cons (Label(u). i))
    FINDALLPREIMAGE(Child(u,i). ChildEdge(u,i). v))
result
return result

Note that performing any list operation on the exception value ⊥ returns ⊥ once again.
Bibliography


81
Bibliography


[29] C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj. "SCR: A Toolset for Specifying and 
Analyzing Software Requirements". In Proc. 10th International Computer 


[33] M. Z. Kwiatkowska, G. Norman, and J. Sproston. "Symbolic Model Checking of 
Probabilistic Timed Automata Using Backwards Reachability". Technical Report 

[34] M.Z. Kwiatkowska, G. Norman, D.A. Parker, and R. Segala. "Symbolic Model 
Checking of Probabilistic Processes Using MTBDDs and the Kronecker representation". In Proceedings of TACAS 2000, number 1587 in Lecture Notes in Computer 


