Infinitary Relations and Their Representation

Laurent Mauborgne [1]

DI, École Normale Supérieure, 45 rue d'Ulm, 75 230 Paris cedex 05, France
Email: Laurent.Mauborgne@ens.fr
WWW home page: `http://www.di.ens.fr/~mauborgn/`

Abstract

This papers presents extensions of Binary Decision Diagrams to represent some infinitary relations (coded as infinite boolean functions). Four classes of infinitary relations are presented, and their representations are discussed. The widest class is closed under all boolean operations. The three others give rise to a canonical representation, which, when restricted to finite relations, are the classical BDDs. The paper also gives new insights in to the notion of variables and the possibility of sharing variables that can be of interest in the case of finite relations.

Key words: BDD. Infinite Functions. Model Checking. Static Analysis.

1 Introduction

Binary Decision Diagrams (BDDs) were first introduced by Randal E. Bryant in [1]. They turned out to be very useful in many areas where manipulation of boolean functions —or equivalently finite relations— was needed. They allowed a real breakthrough of model checking [2], they have been used successfully in artificial intelligence [3] and in program analysis [4–7].

One limitation of BDDs and its variants is that they can only represent finite relations. Indeed, it induces a well-known and quite annoying restriction on model checking, and it restrains its use in program analysis. This paper explores the possibility of extending BDDs so that they can also represent infinitary relations. This extension will allow the model checking of some infinite state systems or unbounded parametric systems, or the static analysis of the

[1] This work was supported by the RTD project IST-1999-20527 "DAEDALUS" of the european FP5 program.

behavior of infinite systems, where the expression of infinite properties such as fairness is necessary.

During the exploration of such extensions, we will define new classes of infinitary relations (which imply languages of infinite words). For those classes, we will propose two efficient extensions of BDDs: Open Binary Decision Graphs (and dually Closed Binary Decision Graphs), and the more powerful Binary Decision Graphs (BDGs). BDGs correspond to the class of ω-deterministic relations.

The goal of these data structures is to represent infinitary relations with variables (BDDs represent finite relations with variables) in an efficient way. As efficiency is a concept relative to the use of the data structure, we need to tell more about it. What we need in static analysis (and model checking) is an expressive data structure (to give more precise results) and to perform inclusion testing (so emptiness testing should be fast). Other operations are application dependant, but most of the time, they can benefit from a good memoizing, which means that we can use a good equality testing. The data structures we propose are canonical, which means equality testing and emptiness testing in constant time. There is a hidden complexity, though, due to the fact that we need to compute the memory location for each new data (to recognize whether it had been encountered before), but this is computed in an incremental way.

After a presentation of our notations, sections 3 and 4 present the main ideas that allow this extension: the first idea is the possibility of sharing variables for different entries, while preserving a sound elimination of redundant nodes. The second idea is the possibility of looping in the representation (thus the term of Binary Decision Graph instead of Binary Decision Diagram), while preserving the uniqueness of the representation. Section 5 presents the first class of functions and their representation. This first class is simple and closed under union and intersection. The next section presents a way of representing more complicated infinite functions by giving a new semantics to the graphs. The widest class of functions is defined in section 7. This class is very expressive and with good algebraic properties, but the representation as an extension of BDDs does not seem efficient enough. For a better efficiency, a restriction of this class is defined in section 8. These functions can be represented by extensions of BDDs, in the sense that if the same representation is applied to finite functions we obtain classical BDDs. This class is quite powerful but not closed under all boolean operations. We propose the use of results on approximation properties of this class that can easily be exploited in abstract interpretation [8,9], a direction that is very promising for its usefulness in program analysis.

2 Definitions and Notations

2.1 Relations and Boolean Functions

Definition 1 *A* relation $\mathcal{R}$ *over the family* $(E_i)_{i\in I}$ *of finite sets is a subset of the cartesian product* $\bigotimes_{i\in I} E_i$ *of the sets.*

An element of $\bigotimes_{i\in I} E_i$ is called a *vector*, thus a relation is a set of vectors of the same length. If I is finite of cardinality n, the relation is said to be *n-ary*. If I is infinite, the relation is called *infinitary*. In this article we will restrict infinitary relations to countable infinitary relations, i.e. I will be countable.

Relations over families of finite sets can be encoded into relations over families of boolean sets (sets of cardinality 2). We just have to replace each set E_i of cardinality n_i by $\lceil \log_2(n_i) \rceil$ boolean sets. This binary encoding is used in [10] to represent finite relations with BDDs. In order to simplify our problems, we will just consider relations over families of boolean sets, keeping in mind that we can always come back to the general case.

Let $\mathcal{B} \stackrel{\text{def}}{=} \{\texttt{true}, \texttt{false}\}$ be the set of boolean values. $\mathcal{B}^n$ denotes the set of boolean vectors of length n. A finite boolean function is a function of $\mathcal{B}^n \rightarrow \mathcal{B}$. The set of infinite boolean vectors will be written $\mathcal{B}^\omega$. An infinite boolean function is a function of $\mathcal{B}^\omega \rightarrow \mathcal{B}$.

A boolean function $\mathcal{F}$ is entirely characterized by a set of vectors, which is defined as $\{u \mid \mathcal{F}(u) = \texttt{true}\}$. This set of vectors of same length n can be seen as an n-ary relation between elements of $\mathcal{B}$. If the vectors are infinite, this set forms an infinitary relation. Thus, we will indifferently write $u \in \mathcal{F}$ or $\mathcal{F}(u) = \texttt{true}$, and consider relations as boolean functions and vice versa. If for all vectors $\mathcal{F}$ is false, we will write $\mathcal{F} = \emptyset$.

2.2 Entries

An important aspect of relations is the indexes in the cartesian product. The corresponding concept in functions is the rank of the parameters used to compute the value of the function. We use the term "entry" for this concept. In the case of functions, this concept is often mixed up with the variables used in the description of the functions (as in $f(x, y, ..) = \ldots$). What we want here is to distinguish between the variables of the functions and their position in the function, which we call its *entries*. If $\mathcal{F} : \mathcal{B}^n \rightarrow \mathcal{B}$, then the entries of $\mathcal{F}$ are the integers between 0 and $n-1$. If $\mathcal{F} : \mathcal{B}^\omega \rightarrow \mathcal{B}$, then the entries of $\mathcal{F}$ are $\mathbb{N}$, the set of all natural numbers.

Example 2 *let $\mathcal{F}$ be the boolean function defined as $\mathcal{F}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = (\boldsymbol{y} \wedge \boldsymbol{z}) \vee (\neg\boldsymbol{x} \wedge \neg\boldsymbol{y} \wedge \neg\boldsymbol{z})$. The function $\mathcal{F}$ is in $\mathcal{B}^3 \to \mathcal{B}$. Thus its entries are 0, 1 and 2, and in the definition shown above, the variable $\boldsymbol{x}$ is associated with the entry 0 (is at position 0), $\boldsymbol{y}$ with 1 and $\boldsymbol{z}$ with 2.*

We write $u_{(i)}$ for the i^{th} component of u. Given a set I of entries, $u_{(I)}$ denotes the subvector of u with I as its set of entries. The *restriction* of $\mathcal{F}$ according to one of its entries i and to the boolean value b is denoted $\mathcal{F}|_{i \leftarrow b}$ and is defined as the set of vectors: $\left\{u \mid \exists v \in \mathcal{F},\ v_{(i)} = b \text{ and } v_{(\{j \mid j \neq i\})} = u\right\}$. A special case is when $i = 0$. In this case, we simply write $\mathcal{F}(b)$.

2.3 Vectors and Words

It is sometimes convenient to consider a boolean vector as a word over $\mathcal{B}^*$ or $\mathcal{B}^\omega$. It allows the use of concatenation of vectors. If u is a finite vector and v a vector, the vector $u.v$ corresponds to the concatenation of the words equivalent to the vectors u and v. The size of a vector u is written $|u|$. The empty word is denoted ε. We define formally the notation $\mathcal{F}(u)$:

$$\mathcal{F}(u) \stackrel{\text{def}}{=} \{v \mid u.v \in \mathcal{F}\} \text{ if } \mathcal{F} : \mathcal{B}^\omega \to \mathcal{B} \text{ or } \mathcal{F} : \mathcal{B}^n \to \mathcal{B} \text{ and } |u| < n$$

Note that this definition is consistent with the notation $\mathcal{F}(b)$ above: if $|u| = 1$, $w = u.v$ means that $w_{(0)} = u$ and $w_{(\{i \mid i > 0\})} = v$. Thus $\mathcal{F}(u) = \mathcal{F}|_{0 \leftarrow u}$.

We extend the concatenation to sets of vectors: for example, if $\mathcal{F}$ is a set of vectors and u a vector, $u.\mathcal{F} \stackrel{\text{def}}{=} \{u.v \mid v \in \mathcal{F}\}$.

This point of view allows the comparison between classes of infinitary relations and classes of ω-languages. Although those classes look very similar, they are not exactly the same, because of the presence of entries in functions and relations. It is important for such datas that we have a way of naming and accessing them. If we enriched word languages with a structure to name entries, it would be possible to represent infinitary relations using Büchi automata [11]. A problem of this representation is the well known lack of efficiency [12]. Because this class has been well studied, we do not try to go beyond it but we will define smaller classes with better representations for practical use.

2.4 Notations

We adopt the following conventions:

a, b, c represent boolean values,

$\mathcal{E}, \mathcal{F}, \mathcal{G}$ represent boolean functions, or equivalently relations,
u, v, w represent vectors (finite or infinite). In a context where we have infinite vectors as described below, represent finite vectors,
α, β, γ represent infinite vectors,
$\mathtt{x}, \mathtt{y}, \mathtt{z}$ represent variables (or entry names),
r, s, t represent binary trees,
i, j, k, n represent natural numbers.

In the description of vectors, to reduce the size of the description, we will write 0 for `false`, and 1 for `true`.

3 Entry Names

As mentioned earlier, it is important for relations and functions to be able to access easily the different entries, so that we can have efficient restrictions or computations of the result. A common way of doing so is by naming the entries, using variables.

Definition 3 (Named Function) *A* named function $\mathcal{F}$ *is a function associated with a naming of its entries. This function, mapping entries to names, will be denoted* $name_{\mathcal{F}}$.

Example 2 (continued). *If* $\mathcal{F}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \ldots$ *is our named function, then* $name_{\mathcal{F}}(0) = \boldsymbol{x}$, $name_{\mathcal{F}}(1) = \boldsymbol{y}$ *and* $name_{\mathcal{F}}(2) = \boldsymbol{z}$.

In classical BDDs, boolean functions are described by boolean expressions using variables corresponding to the different entries of the functions. The Binary Decision Diagrams are ordered, so that the ordering imposed on the variables follows the order of the entries of the function. The variable of rank i is called the name of the entry i. In many cases, the variables correspond to some entities related by the boolean function. A consequence is that we can have the same information while changing the ordering on the variables and the boolean functions that bind them (so that a given entry always corresponds to the same variable, and the ordering on the variables is the same as the entries ordering). Different optimizations follow that depend on the choice of this ordering [13,14].

In the case of infinite functions, we cannot assign a different variable to all the entries of the boolean function, because we want a finite representation. The idea is that the set of variables associated with a given function is finite and to achieve that, some entries can share the same name. However, not every entry can share the same name: to share the same name, two entries must be in a sense equivalent.

This idea is necessary to have a representation of infinite functions with named entries. But, as shown in example 12 (page 10 and next), the equivalence of entries can even be used in the representation of finite functions with named entries, such as BDDs. It might be easier to think of the entry names in BDDs to follow this sections. In BDDs, entry names are the variables which are the labels of the nodes. What we propose to do here is to rename some of those labels, in a way compatible with the decision semantics of BDDs. Note that this would not change the size of the BDD.

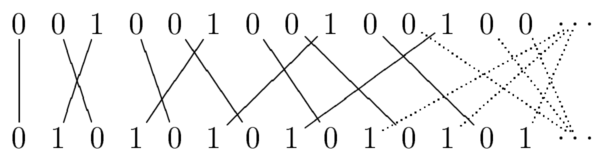
A permutation is a bijection of $\mathbb{N} \rightarrow \mathbb{N}$. If $I \subset \mathbb{N}$, a permutation of the entries in I is a permutation σ such that $\forall i \notin I$, $\sigma(i) = i$. A permutation σ defines a function from vectors to vectors, $\overrightarrow{\sigma}$, defined by : $\overrightarrow{\sigma}(u)_{(i)} = u_{(\sigma(i))}$ for all i entry of u.

Definition 4 (Equivalent entries) *Let $\mathcal{F}$ be a boolean function. The entries contained in the set $I \subset \mathbb{N}$ are equivalent if and only if for any permutation σ of the entries in I, for any vector $u \in dom(\mathcal{F})$, $\mathcal{F}(u) = \mathcal{F}(\overrightarrow{\sigma}(u))$*

There are two ideas underlying the definition of equivalent entries: the restriction according to any equivalent entry is the same, so $\mathcal{F}|_{\mathtt{x} \leftarrow b}$, where `x` is an entry name, is not ambiguous (see property 7); and whatever the order in which we read the equivalent entries, the function is the same. The following example shows that this property imposes that we allow infinite permutations.

Example 5 *Consider the infinite function that is true on any infinite vector containing two consecutive 0's infinitely many times. This function contains $(001)^\omega$ but not $(01)^\omega$. If we only considered finite permutations, that is permutations generated from finite exchange of entries, then all entries of this function are equivalent. But there is an infinite permutation that transforms $(001)^\omega$ into $(01)^\omega$:*

$$\sigma(3 \times k) = 4 \times k$$
$$\sigma(3 \times k + 1) = 4 \times k + 2$$
$$\sigma(3 \times k + 2) = 2 \times k + 1$$



The meaning of this substitution is that, if a function with all entries equivalent contains $(001)^\omega$ then there is a way of giving the values of $(01)^\omega$ such that it is accepted by the function. Concerning our function, because $\mathcal{F}((001)^\omega) \neq \mathcal{F}((01)^\omega)$, we must forbid the equivalence of all entries.

As an immediate consequence, we have the following properties for functions $\mathcal{F}$ where all entries are equivalent:

Proposition 6 *Let $\mathcal{F} : \mathcal{B}^\omega \to \mathcal{B}$, such that the entries of $\mathcal{F}$ are all equivalent. Let v be a word, b a boolean value in v, α be a word where the boolean value a appears infinitely often. Then $v^\omega \in \mathcal{F}$ if and only if $(vb)^\omega \in \mathcal{F}$ and $\alpha \in \mathcal{F}$ if and only if $b.\alpha \in \mathcal{F}$.*

PROOF. In both cases, we have an infinite permutation of the entries that transforms the first vector into the other one. In the second case, we just have to shift the a's of $a.\alpha$ to the right, each a going to the entry of the next one. In the first case, we keep shifting by one more b for each $v.b$. $\square$

The following property proves that the notation $\mathcal{F}|_{\mathtt{X}\leftarrow b}$ is not ambiguous:

Proposition 7 *Let I be a set of equivalent entries of the function $\mathcal{F}$. Whatever $i, j \in I$ and $b \in \mathcal{B}$, $\mathcal{F}|_{i\leftarrow b} = \mathcal{F}|_{j\leftarrow b}$.*

PROOF. By definition of the restriction, $\mathcal{F}|_{i\leftarrow b}(u) = \exists v \in \mathcal{F},\ v_{(i)} = b$ and $v_{(\{\,k\,|\,k\neq i\})} = u$. If $\mathcal{F}|_{i\leftarrow b}(u) = \mathtt{true}$, let v be the vector defined above. Let σ be the permutation that exchanges i and j. By equivalence of i and j in $\mathcal{F}$, $\overrightarrow{\sigma}(v) \in \mathcal{F}$. Moreover, by the action of the permutation, $\overrightarrow{\sigma}(v)_{(j)} = b$ and $\overrightarrow{\sigma}(v)_{(\{\,k\,|\,k\neq j\})} = u$, so $\mathcal{F}|_{j\leftarrow b}(u) = \mathtt{true}$. The converse is true by symmetry of the property, which proves the equality of $\mathcal{F}|_{i\leftarrow b}$ and $\mathcal{F}|_{j\leftarrow b}$. $\square$

3.2 Equivalent Vectors of Entries

In order to be able to represent a wider class of infinite functions (while keeping the number of entry names finite), we extend the notion of equivalent entries to equivalent vectors of entries. We just consider as one entry a whole set of entries of the form $\{i \in \mathbb{N} \mid k \leq i < k+n\}$. It will allow the iteration over whole vectors of entries. The set of equivalent entries is described by a set I of indexes and a length n such that $\forall k \in I$, $\forall i$ such that $k < i < k+n$, $i \notin I$. A substitution σ over such a set is such that $\forall k \in I$, $\sigma(k) \in I$, and $\forall i < n$,

$\sigma(k+i) = \sigma(k) + i$. For all other numbers j, $\sigma(j) = j$.

Definition 8 (Equivalent Vectors of Entries) *Let $\mathcal{F}$ be a boolean function. The vectors of entries contained in the set I with length n are equivalent if and only if for any permutation σ of the entries in I, for any vector $u \in dom(\mathcal{F})$, $\mathcal{F}(u) = \mathcal{F}(\overrightarrow{\sigma}(u))$*

Two entries can have the same name if and only if they are at the same position in a set of equivalent vectors of entries. In the remainder, when considering named functions, and to simplify the presentation and the proofs, we will only consider simple equivalent entries, but the results extend easily to equivalent vectors of entries.

3.3 Equivalent Entries and Redundant Choices

Redundant choices are used in BDDs to reduce the size of the representation. The good news is that giving the same name to equivalent entries is compatible with the elimination of redundant choices. There is a redundant choice at a subvector u of $\mathcal{F}$ if and only if $\mathcal{F}(u.0) = \mathcal{F}(u.1)$.

Theorem 9 *Let $\mathcal{F}$ be a named boolean function, and u be a vector such that $\mathcal{F}(u.0) = \mathcal{F}(u.1)$. Then, whatever v such that $name_{\mathcal{F}}(|u.v|) = name_{\mathcal{F}}(|u|)$, $\mathcal{F}(u.v.0) = \mathcal{F}(u.v.1)$.*

PROOF. $v = a.w$. We have $\mathcal{F}(u.a.w.0) = \mathcal{F}(u.0.w.a)$ because of the equivalence of the entries. $\mathcal{F}(u.0.w.a) = \mathcal{F}(u.1.w.a)$ by redundancy of the choice, and $\mathcal{F}(u.1.w.a) = \mathcal{F}(u.a.w.1)$ by equivalence of the entries. Thus $\mathcal{F}(u.v.0) = \mathcal{F}(u.v.1)$. $\square$

3.4 Periodicity of the Entries

In order to be finitely representable, we impose some regularity to the entry names. The entry names are said to be *periodic* if and only if there is a period k on the entry names, that is, for all i, the name of $i + k$ is the same as the name i. The entry names are said to be *ultimately periodic* if, after some point, they are periodic.

Definition 10 *A named function $\mathcal{F}$ is said to have* ultimately periodic entry names *iff there is a period k and an entry j such that for all entry $i > j$, $name_{\mathcal{F}}(i) = name_{\mathcal{F}}(i + k)$.*

In the remainder, we will define new classes of infinitary relations which could be represented by extensions of BDDs. A direct consequence of this discussion over entries and entry names is that for each such class of relations, the entry names will have to be ultimately periodic. It is a strict restriction, in particular for ω-regular languages of Büchi.

Proposition 11 *The class of ω-regular languages of Büchi such that there exists a named function with ultimately periodic entries of domain the same set of words is a strict subset of the class of ω-regular languages of Büchi.*

PROOF. The set $\{0, 11\}^\omega$ is an ω-regular language. Suppose there is a named function with ultimately periodic entries which is `true` on exactly that set. As the entries are ultimately periodic, there must be at least two equivalent entries $i < j$. Then $0^j 110^\omega$ is in the function, so $0^i 10^{j-i} 10^\omega$ should be in the function too, because of the equivalence of entries i and j. It means that i and j are not equivalent. □

4 Decision Trees

4.1 Finite Decision Trees

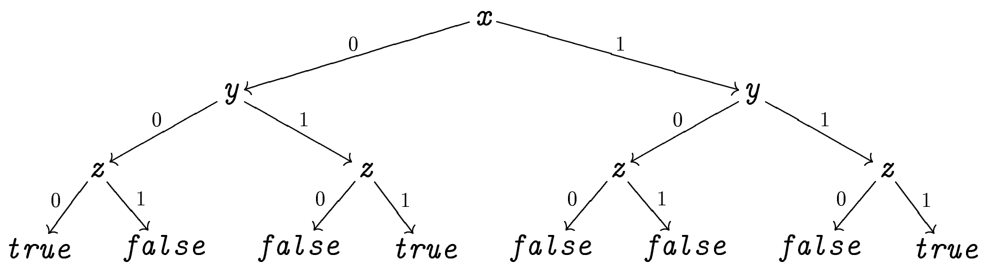
BDDs are based on decision trees. A decision tree is a structured representation based on Shannon's expansion theorem: $\mathcal{F} = 0.\mathcal{F}(0) \cup 1.\mathcal{F}(1)$. This observation is the basis of a decision procedure: to know whether a given vector is in a relation, we look at its first value. If it is a 0, we iterate the process on the rest of the vector and $\mathcal{F}(0)$, and if it is a 1, we iterate on the rest of the vector and $\mathcal{F}(1)$. This procedure can be represented by a binary tree labeled by the entry names, and with either `true` or `false` at the leaves.

The construction of a decision tree for a finite function is classical, but we will rephrase it in our terminology. This will allow us to show how entries and entry names are used, and then how the distinction between those two concepts (which are usually mixed in the term "variable") can give more freedom in the construction of those decision trees.

We define a labeled binary tree t as a partial function of $\{0, 1\}^* \to L$ where L is the set of labels, and such that whatever $u.v \in dom(t)$, $u \in dom(t)$. The subtree of t rooted at u is the tree denoted $t_{[u]}$ of domain $\{v \mid u.v \in dom(t)\}$, and defined as $t_{[u]}(v) \stackrel{\text{def}}{=} t(u.v)$. The decision tree defined by a named boolean function $\mathcal{F} : \mathcal{B}^n \to \mathcal{B}$ is the binary tree of domain $\bigcup_{k \leq n} \{0, 1\}^k$, such that if

$|v| = n$, then $t(v) = \mathcal{F}(v)$, and if $|v| < n$, $t(v) = \mathrm{name}_{\mathcal{F}}(|v|)$. Note that this definition does not impose that all entry names be different.

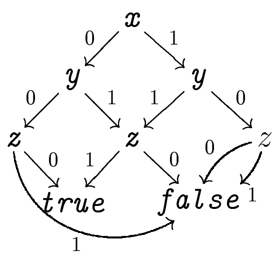
Example 12 *Let* $\mathcal{F} = \{000, 011, 111\}$. *If we associate the variables* $\boldsymbol{x}$ *to entry 0,* $\boldsymbol{y}$ *to entry 1 and* $\boldsymbol{z}$ *to entry 2, then* $\mathcal{F}$ *can be described by the formula:* $(\boldsymbol{y} \wedge \boldsymbol{z}) \vee (\neg\boldsymbol{x} \wedge \neg\boldsymbol{y} \wedge \neg\boldsymbol{z})$. *The decision tree for* $\mathcal{F}$ *is:*

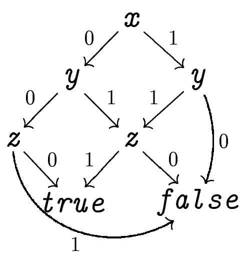


4.2 Semantics of the Decision Trees

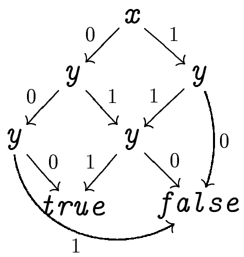
The decision tree of a named boolean function is used as a guide for a decision process that decides the value of the function on a given vector. If t is the decision tree associated with the named function $\mathcal{F}$, then to decide whether the vector u is in $\mathcal{F}$, we "read" the first value of u. Say $u = b.v$. If b is a 0, we iterate on v, $t_{[0]}$, and if it is a 1, we iterate on v, $t_{[1]}$. The entire decision process goes through the tree, following the path defined by u, and the result of the decision process is the value of the leaf, $t(u)$. Binary Decision Diagrams are based on two remarks: first we can represent any tree in a form where equivalent subtrees are shared (a directed acyclic graph), and second if a choice is redundant, then we can "jump it". The second remark modifies slightly the decision process: we must have separate information on the entries of a function. Without the elimination of redundant nodes, the entry names labeling the tree would be of no use. But if we allow the elimination of redundant nodes, then they allow to keep track of the current entry that is read from u. If it is "before" the entry named $t(\varepsilon)$, we can skip the first value of u and iterate on v, t. It means also that we need a way of representing $\mathrm{name}_{\mathcal{F}}$. In classical BDDs, where all entry names are different, we can represent $\mathrm{name}_{\mathcal{F}}$ by an ordering of the entry names: $\mathrm{name}_{\mathcal{F}}(i)$ is the i^{th} entry name in the ordering. If we allow to have the same entry names at different entries, then we can use a sequence of entry names, and $\mathrm{name}_{\mathcal{F}}(i)$ is the i^{th} entry name in the sequence.

Example 12 (continued). *The two steps of sharing equivalent subtrees and eliminating redundant nodes give the following diagrams:*





The decision process on the vector 101 *can reach* `false` *after reading the first 1 and the first 0. The entry names of the BDD are represented by* $\boldsymbol{x} < \boldsymbol{y} < \boldsymbol{z}$*, or equivalently by* $\boldsymbol{xyz}$*. If we realize that the entry 1 and the entry 2 are equivalent, we can give the same name* $\boldsymbol{y}$ *to both entries. Then the BDD becomes:*



with entry names described as $\boldsymbol{xyy}$*. Note that this cannot change the size of the BDD. There can be an interest in it, though, even for finite functions. To compute a restriction, one needs to go through the diagram up to the nodes labeled by the corresponding name, so the farther the name in the diagram, the more complex the restriction. It is easy to see that this description will lead to a more efficient algorithm to compute* $\mathcal{F}|_{2\leftarrow 0}$ *which corresponds to* $\mathcal{F}|_{\boldsymbol{z}\leftarrow 0}$ *with the entry names* $\boldsymbol{xyz}$*, and to* $\mathcal{F}|_{\boldsymbol{y}\leftarrow 0}$ *with the entry names* $\boldsymbol{xyy}$*. Concerning the meaning of the diagram, note that we don't test* $\boldsymbol{y}$ *twice, but we test the entries* 1 *and* 2*, and the fact that they have the same name means that we can test them in any order and obtain the same result.*

It is an established fact [1] [2], that given a boolean function and a naming of the entries with all names different, such a representation is unique, leading to trivial equivalence testing. From Theorem 9, we can add that this representation is still unique if the naming of the entries respects the equivalences of the entries.

[2] Although we rephrase it with entry names instead of variables.

To extend this definition to infinite boolean functions, there are two problems: for all $\alpha \in \mathcal{F}$, $\alpha \notin dom(t)$, because binary tree domains are limited to finite words. This is the problem of the *infinite behavior* of the function. We can represent $t(v)$ for all v a prefix of a vector in $\mathcal{F}$, but then the tree is infinite. This is the second problem, treated in this section: how to represent an infinite tree.

As we have seen, a BDD is a decision tree on which we have performed two operations: first the sharing of equivalent subtrees, second the elimination of redundant choices. In fact, following this process would be too inefficient, and when manipulating BDDs, these operations are performed incrementally: each time we build a tree $\underset{t\quad t}{\overset{x}{\swarrow\searrow}}$, we return t, and each time we build another tree $\underset{t_0\quad t_1}{\overset{x}{\swarrow\searrow}}$, we first look if the tree has already been encountered, through a hash table for example, and if it is the case, we return the tree already encountered, if not we add it in the table.

The same operations, albeit a little more complex, can be performed to represent an infinite tree with maximal sharing of its subtrees [15]. First we only represent *regular* trees, that is trees with a finite number of distinct subtrees. The only difference with finite trees, which are represented by directed acyclic graphs, is that infinite trees are represented by directed graphs that contain cycles. The added complexity introduced by the cycles is not intractable, and efficient incremental algorithms can be devised. The ideas are the following: when we are not in a cycle, the algorithm is the same as in the finite case. When we isolate a strongly connected subgraph (a “cycle”), we first see if this cycle is not the unfolding of another cycle that is reachable from the subgraph. If it is the case, we return this other cycle (we fold the subgraph on the cycle). If not, we reduce the subgraph to an equivalent one with maximal sharing, and then we compute unique keys for the subgraph, so that we can see if it had already been encountered, or so that we can recognize it in the future. We have one key for each node of the subgraph. The detailed algorithms and their proofs can be found in [16].

Examples of infinite trees represented this way will be displayed in the next sections. The trees will be progressively enriched so that the BDGs represent wider classes of infinite functions.

5 Simple Infinite Behaviors

Using the ideas of the last two sections, we can extend BDDs to represent classes of infinitary relations with simple infinite behavior. This extension consists in allowing the sharing of entry names (to have an infinite number of entries represented by a finite number of entry names), and in allowing loops in the diagrams. We need to define the classes of boolean functions which can be represented by these extensions of BDDs. The first class we define is a superset of those classes derived from the necessity that the decision tree be representable, that is regular.

Definition 13 *Let $\mathcal{F}$ be a named boolean function. $\mathcal{F}$ is said to be* prefix regular *if and only if the number of distinct $\mathcal{F}(u)$ (as defined in section 2.3) is finite and its entry names are ultimately periodic.*

Because we will have only one possible representation for a given function, and $\mathcal{F}(u)$ corresponds to $t(u)$ if t represents $\mathcal{F}$, it means that t is regular.

Note that prefix regularity of a funtion $\mathcal{F}$ does not necessarily means that its domain is ω-regular: suppose we have a non ω-regular language $\mathcal{L}$. Then $\mathcal{F} = \{0,1\}^*.\mathcal{L}$ is non ω-regular either, but all $\mathcal{F}(u) = \mathcal{F}$.

5.1 Open Functions and Closed Functions

We mentioned earlier that the representation of the decision tree for infinite functions presented two problems: the representation of each partial evaluation of the function (solved by a regular tree), and the problem of the decision process which must be infinite and cannot be represented in general by a mere regular tree. The idea, to solve this infinite behavior problem simply, is to give a uniform meaning to the possible infinite decision processes, that is the infinite loops in the decision tree. And to keep as close as possible to classical BDDs, if the decision process is finite, then the result should be the same as with BDDs. In particular, if we arrive at a `false` node after reading the start u of the vector, then there is no need to go further since no vector starting by u will be in the relation ($\mathcal{F}(u) = \emptyset$), and if we come to a `true`, then any vector starting by u will be in this relation ($\mathcal{F}(u) = \mathcal{B}^\omega$).

We can define two classes of boolean functions, corresponding to two possible meanings for infinite loopings (we never reach `true` nor `false`), namely exclusive (for open functions) and inclusive (for closed functions).

Definition 14 (Open Function) *Let $\mathcal{F} : \mathcal{B}^\omega \to \mathcal{B}$. The function $\mathcal{F}$ is said* *to be* open *if and only if $\mathcal{F}$ is prefix regular and:*

$$\forall \alpha \in \mathcal{F},\ \exists u,\ \beta \text{ such that } \alpha = u.\beta \text{ and } \mathcal{F}(u) = \mathcal{B}^{\omega}$$

Recall that $\mathcal{F}(u) = \mathcal{B}^{\omega}$ means that whatever γ, $\mathcal{F}(u.\gamma) =$ `true`. It means that the only vectors in the relation will lead finitely to a `true`. The dual definition is:

Definition 15 (Closed Function) *Let $\mathcal{F} : \mathcal{B}^{\omega} \to \mathcal{B}$. The function $\mathcal{F}$ is said to be* closed *if and only if $\mathcal{F}$ is prefix regular and:*

$$\forall \alpha \notin \mathcal{F},\ \exists u,\ \beta \text{ such that } \alpha = u.\beta \text{ and } \mathcal{F}(u) = \emptyset$$

As those two definitions are dual, we will mainly describe the representation and properties of one of them, say open functions. The results on open functions can be translated for closed functions by exchanging the roles of `true` and `false`. The choice between one class of relations or the other will depend on the particular applications, but we cannot mix them.

5.2 Open BDGs

Because we chose that cycling in the decision process is rejecting, there is one new source of non-uniqueness that is not taken care of by simply sharing every subtree of the decision tree. We must also replace by `false` every cycle from which no `true` is reachable. This is easily performed while treating cycles in the representation of regular trees.

We define the notion of decision tree representing an open function:

Definition 16 *A decision tree* `dt` *represents a named open function $\mathcal{F}$ if and only if the labels of the inner nodes of* `dt` *are* `dt`$(u) = name_{\mathcal{F}}(|u|)$ *and $\forall \alpha \in \mathcal{F}$, $\exists u$, β such that $\alpha = u.\beta$ and* `dt`$(u) =$`true`*, and $\forall u$ such that* `dt`$(u) =$`true`*, $\mathcal{F}(u) = \mathcal{B}^{\omega}$.*

Now, we can prove that we have a unique representation for open functions:

Theorem 17 *There is exactly one decision tree* `dt` *representing the named open function $\mathcal{F}$ such that for all r subtree of* `dt`*, either $r =$* `false` *or at least one leaf of r is* `true`*, and if r is finite, it is* `true`*,* `false` *or both leaves appear.*

The last condition just corresponds to the elimination of redundant nodes before `true` and `false`, because we could have any number of such nodes.

PROOF. It is obvious from definition 16 that there exists at least one decision tree `dt` representing $\mathcal{F}$. Suppose there is another decision tree `dt'` representing also $\mathcal{F}$ with the same conditions, then there must be a u such that one of the trees, say `dt` is labeled by `true` or `false` at u, and `dt'` is labeled by an entry name. Let r be the subtree of `dt'` at u. If $\mathtt{dt}(u) = $`true`, then $\mathcal{F}(u) = \mathcal{B}^\omega$, so r cannot contain any loop (which would exclude a vector), and the only possible leaves are `true`. This contradicts the second condition. If $\mathtt{dt}(u) = $`false`, then, in the same way, no leaf of r can be `true`, so, by the first condition, r must be `false`. $\square$

The *open BDG* for a function is defined as the the unique representation (see [16] for a proof) of this decision tree, with the elimination of the redundant nodes. Open BDGs are indeed extensions of BDDs, because they have the same unique incremental representation principle, and if we apply the open BDG representation to finite functions (which are both open and closed), we obtain the classical BDDs.

Example 18 *The function $\mathcal{F}$ defined in example 12 can be extended to the function $\mathcal{G} : \mathcal{B}^\omega \to \mathcal{B}$, defined as: $\mathcal{G}(u.\alpha) = $ **`true`** if $u \in \mathcal{F}$. Then $\mathcal{G}$ is represented with the same diagram as $\mathcal{F}$, with an additional entry name z', and the entry names are $\boldsymbol{xyy}(\boldsymbol{z'})^\omega$.*

Example 19 *Let $\mathcal{F}$ be **`true`** on α if and only if α contains at least one 1. All entries of $\mathcal{F}$ are equivalent, so its entry names can be described as $\boldsymbol{x}^\omega$. Since $\mathcal{F}$ is open, it can be represented by the following graph:*

$$\begin{tikzcd} \boldsymbol{x} \arrow[loop left, "0"] \arrow[dr, "1"] & \\ & \mathtt{true} \end{tikzcd}$$

*Note that this diagram would not be a correct representation for a closed function, because there is no path leading to **`false`** (the dual of **`true`**).*

5.3 Boolean Operators

We will write $\mathcal{F} \wedge \mathcal{G}$ to denote the intersection of two relations $\mathcal{F}$ and $\mathcal{G}$, and $\mathcal{F} \vee \mathcal{G}$ for their union.

Theorem 20 *Let $\mathcal{F}$ and $\mathcal{G}$ be two open functions. Then the functions $\mathcal{F} \wedge \mathcal{G}$ and $\mathcal{F} \vee \mathcal{G}$ are open. Moreover, if $(\mathcal{F}_i)_{i\in\mathbb{N}}$ is a family of open functions, then $\bigvee_{i\in\mathbb{N}} \mathcal{F}_i$ is an open function.*

$\mathcal{F} \wedge \mathcal{G}(\alpha) \stackrel{\text{def}}{=} \mathcal{F}(\alpha) \wedge \mathcal{G}(\alpha)$, and $\mathcal{F} \vee \mathcal{G}(\alpha) \stackrel{\text{def}}{=} \mathcal{F}(\alpha) \vee \mathcal{G}(\alpha)$. So, if we consider $\mathcal{F}$ and $\mathcal{G}$ as sets of vectors, $\mathcal{F} \wedge \mathcal{G}$ is the intersection of $\mathcal{F}$ and $\mathcal{G}$, and $\mathcal{F} \vee \mathcal{G}$ is the union of $\mathcal{F}$ and $\mathcal{G}$.

PROOF. Let $\alpha \in \mathcal{F} \vee \mathcal{G}$. There is a u such that $\alpha = u.\beta$, and either $\mathcal{F}(u) = \mathcal{B}^\omega$ or $\mathcal{G}(u) = \mathcal{B}^\omega$. In any case, $\mathcal{F} \vee \mathcal{G}(u) = \mathcal{B}^\omega$. If $\alpha \in \mathcal{F} \wedge \mathcal{G}$, there is u and v such that $\alpha = u.\beta$, $\alpha = v.\gamma$, and $\mathcal{F}(u) = \mathcal{B}^\omega$ and $\mathcal{G}(v) = \mathcal{B}^\omega$. If $|u| \leq |v|$, then $v = u.w$. So $\mathcal{F}(v) = \mathcal{B}^\omega$. So, $\mathcal{F} \wedge \mathcal{G}(v) = \mathcal{B}^\omega$. If $\alpha \in \bigvee_{i \in \mathbb{N}} \mathcal{F}_i$, then there is a least u prefix of α such that there is a i, $\mathcal{F}_i(u) = \mathcal{B}^\omega$. We have $\bigvee_{i \in \mathbb{N}} \mathcal{F}_i(u) = \mathcal{B}^\omega$. $\square$

Dually, the finite union of closed functions is a closed function, and the infinite intersection of closed functions is a closed function.

Corollary 21 *Whatever the boolean function $\mathcal{F}$, there is a greatest open function contained in $\mathcal{F}$, and there is a least closed function containing $\mathcal{F}$.*

Algorithmically, it is easy to compute the `and` or the `or` of two open functions. The algorithms are the same as in the finite case [17], with the possibility of memoizing [18], except that we must take care of cycles. When a cycle is encountered, that is when we recognize that we already have been through a pair of subtrees (s, t), we build a loop in the resulting tree.

Open functions are not closed under negation: the negation of the function that is true on all vectors containing at least one 1 is the function containing only 0^ω. Such a function is not open (but it is closed, of course), because the only infinite behavior that is possible for an open function is trivial. In order to be more expressive, we introduce more infinite behaviors.

6 More Infinite Behaviors

To allow more infinite behaviors, we need to have more than one kind of loop, so that in some loop it is forbidden to stay forever, and in some others, we can. We introduce a new kind of loop: loops over open functions. This new kind of loop defines a new set of infinite behaviors, defining what we call iterative functions. Iterative functions are functions that start over again and again infinitely often. Thus entry names will have to be periodic.

6.1 Definition

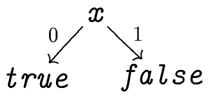
Definition 22 (Iteration) *Let $\mathcal{F} : \mathcal{B}^\omega \to \mathcal{B}$ be a named function. The iteration of $\mathcal{F}$, noted $\Omega(\mathcal{F})$, is defined as the set of vectors α such that there is an* *infinite sequence of vectors* $(u_i)_{i \in \mathbb{N}}$ *and* $\alpha = u_0.u_1 \ldots u_i \ldots$ *and each* u_i *has the minimal length such that:*

(1) $|u_i| > 0$
(2) $\mathcal{F}(u_i) = \mathcal{B}^\omega$
(3) $name_{\mathcal{F}}(|u_i|) = name_{\mathcal{F}}(0)$

Definition 23 (Iterative Function) *Let* $\mathcal{F} : \mathcal{B}^\omega \to \mathcal{B}$ *be a named function.* $\mathcal{F}$ *is said to be* iterative *if and only if the entry names of* $\mathcal{F}$ *are periodic, and there is an open function* $\mathcal{G}$ *such that* $\mathcal{F} = \Omega(\mathcal{G})$.

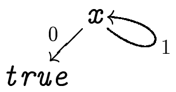
Hence an iterative function is represented by an open function. We will use the decision tree of the open function to represent the iterative function. But in the context of iterative functions, the decision tree will have a different meaning, corresponding to a slightly different decision process. The decision process is the following: we follow the decision tree in the path corresponding to the vector, but when we reach a `true`, we start again at the root of the tree. To be a success, the decision process must start again an infinite number of times.

Example 24 (Safety)



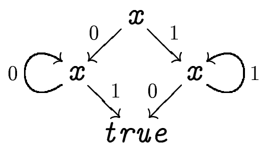
represents the function that is true on 0^ω *only.*

Example 25 (Liveness)



represents the function that is false only on those vectors that end with 1^ω. *That is, the function is true on any vector containing an infinite number of 0's.*

Example 26 (Fairness)



represents the function that is true on any vector containing an infinite number of 0's and 1's.

These examples show that iterative functions can be used to represent a wide variety of infinite behaviors. Note that $\emptyset$ and $\mathcal{B}^\omega$ are at the same time open and iterative. Another remark: the equivalence of the entries restraining the use of shared entry names is only applied to the iterative function, not the open function that represents the iterative function.

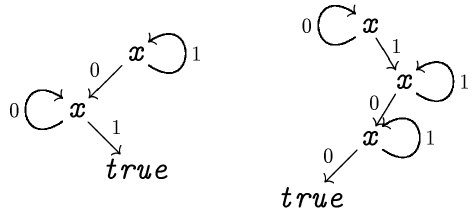
Theorem 27 *An iterative function is prefix regular.*

PROOF. If $\mathcal{F} = \Omega(\mathcal{G})$, g is open, so prefix regular. Let u be a finite vector such that there is a v, $\mathcal{G}(u) = \mathcal{G}(v)$ and $|u| > |v|$. If $\mathcal{G}(u) \neq \mathcal{B}^\omega$ then $\mathcal{F}(u) = \mathcal{F}(v)$. If $\mathcal{G}(u) = \mathcal{B}^\omega$ then there is a u_0 prefix of u and minimal such that $|u_0| > 0$, $\mathcal{G}(u_0) = \mathcal{B}^\omega$ and $\text{name}_\mathcal{G}(|u_0|) = \text{name}_\mathcal{G}(0)$. Let $u = u_0.u_1$. We have $\mathcal{F}(u_0) = \mathcal{F}$, so $\mathcal{F}(u) = \mathcal{F}(u_1)$, with $|u| > |u_1|$. If we take a u of size greater than the number of distinct $\mathcal{G}(u)$, then there exists a v as described above, and so there is a w (either v or u_1 above) such that $\mathcal{F}(u) = \mathcal{F}(w)$ and $|u| > |w|$. So the number of distinct $\mathcal{F}(u)$ is bounded. Thus, $\mathcal{F}$ is prefix regular. $\square$

6.2 Uniqueness

Many possible open functions can represent the same iterative function:

Example 28 *The function that is true on every vector with an infinite number of 0's and 1's could also be represented by the following open function:*



In fact, there is a "best" open function representing a given iterative function. If we always choose this best open function, as the representation of open function is unique, the representation of iterative function is unique too.

Theorem 29 *Let $\mathcal{F}$ be an iterative function. The function $\mathcal{G}$ which is true on the set $\{u.\alpha |\, u^\omega \in \mathcal{F}$ and $\mathcal{F}(u) = \mathcal{F}\}$ is the greatest (for set inclusion) open function such that $\mathcal{F} = \Omega(\mathcal{G})$.*

In order to simplify the proofs, we will suppose that all entries of $\mathcal{F}$ are equivalent, so that there is only one entry name, and we can get rid of the test $\text{name}_\mathcal{F}(|u|) = \text{name}_\mathcal{F}(0)$. To reduce the problem to this case, we can use a function over larger finite sets described by $\mathcal{B}^k$, where k is a period of the entry names of $\mathcal{F}$.

Lemma 30 *Let u be a vector such that $g(u) = \mathcal{B}^\omega$, $|u| > 0$ and for all v prefix of u, $\mathcal{G}(v) \neq \mathcal{B}^\omega$. Then $u^\omega \in \mathcal{F}$ and $\mathcal{F}(u) = \mathcal{F}$.*

PROOF. [Lemma 30] Whatever α, $u.\alpha \in \mathcal{G}$. Because of the minimality of u with respect to the property $\mathcal{G}(u) = \mathcal{B}^\omega$, for each α, there is a v prefix of α such that $(u.v)^\omega \in \mathcal{F}$ and $\mathcal{F}(u.v) = \mathcal{F}$. Let b be a boolean value in u. We choose $\alpha = b^\omega$. There is an l such that $(u.b^l)^\omega \in \mathcal{F}$. Because all entries of $\mathcal{F}$ are equivalent, by Proposition 6, $u^\omega \in \mathcal{F}$.

Let $\alpha \in \mathcal{F}$. The vector α is infinite, so there is a boolean value b that is repeated infinitely often in α. But there is an l such that $\mathcal{F}(u.b^l) = \mathcal{F}$. So $u.b^l.\alpha \in \mathcal{F}$, and by the equivalence of all entries (Proposition 6), $u.\alpha \in \mathcal{F}$, which means that $\alpha \in \mathcal{F}(u)$. Conversely, if $\alpha \in \mathcal{F}(u)$, $u.b^l.\alpha \in \mathcal{F}$, and so $\alpha \in \mathcal{F}$. Thus $\mathcal{F} = \mathcal{F}(u)$.

PROOF. [Theorem 29] $\mathcal{G}$ is an open function, because for any element α of $\mathcal{G}$, there is a u prefix of α such that $\forall\beta$, $u.\beta \in \mathcal{G}$. $\mathcal{F}$ is iterative, so there is an open function $\mathcal{G}'$ such that $\mathcal{F} = \Omega\left(\mathcal{G}'\right)$. $\forall\alpha \in \mathcal{G}'$, there is a u prefix of α such that $\mathcal{G}'(u) = \mathcal{B}^\omega$ and $|u| > 0$. Let u_0 be the least such u. Because $\mathcal{F} = \Omega\left(\mathcal{G}'\right)$, ${u_0}^\omega$ is in $\mathcal{F}$, and $\mathcal{F}(u_0) = \mathcal{F}$. So $\alpha \in \mathcal{G}$, which means that $\mathcal{G}' \subset \mathcal{G}$. To prove the theorem, we just have to prove that $\mathcal{F} = \Omega\left(\mathcal{G}\right)$. We will start by $\mathcal{F} \subset \Omega\left(\mathcal{G}\right)$, then prove $\Omega\left(\mathcal{G}\right) \subset \mathcal{F}$.

Let $\alpha \in \mathcal{F}$. We suppose $\alpha \notin \Omega\left(\mathcal{G}\right)$. If there is a u prefix of α such that $\mathcal{G}(u) = \mathcal{B}^\omega$, let u_0 be the least such u. $\alpha = u_0.\beta$. Then $\beta \notin \Omega\left(\mathcal{G}\right)$, but $\mathcal{F}(u_0) = \mathcal{F}$, so $\beta \in \mathcal{F}$. So we can iterate on β. This iteration is finite because $\alpha \notin \Omega\left(\mathcal{G}\right)$, so we come to a point where there is no u prefix of α such that $\mathcal{G}(u) = \mathcal{B}^\omega$. But $\alpha \in \mathcal{F}$, so there is a u_0 prefix of α such that $\mathcal{G}'(u_0) = \mathcal{B}^\omega$, and $u_0^\omega \in \mathcal{F}$ and $\mathcal{F}(u_0) = \mathcal{F}$, and so $\mathcal{G}(u_0) = \mathcal{B}^\omega$, which contradicts the hypothesis. Thus $\mathcal{F} \subset \Omega\left(\mathcal{G}\right)$.

Let $\alpha \in \Omega\left(\mathcal{G}\right)$. Let $(u_i)_{i\in\mathbb{N}}$ be the sequence of words such that $\mathcal{G}(u_i) = \mathcal{B}^\omega$, $\alpha = u_0.u_1 \ldots u_i \ldots$, $|u_i| > 0$ and u_i minimum. Some letters appear infinitely often in α, and some others appear only finitely often. But there is a finite number of u_i containing the latter ones. Hence there is a permutation of the entries such that the result of the permutation on α is $v.\beta$, where v is the concatenation of all u_i that contain the letters that appear finitely in α (v can be empty), and β is composed of those u_i that contain just letters that appear infinitely in α. By definition of $\Omega\left(\mathcal{G}\right)$, $\beta \in \Omega\left(\mathcal{G}\right)$, and because all entries of $\mathcal{F}$ are equivalent, $v.\beta$ is in $\mathcal{F}$ if and only if α is in $\mathcal{F}$. But whatever u_i, $\mathcal{F}(u_i) = \mathcal{F}$ (see the lemma). So $\mathcal{F}(v) = \mathcal{F}$. And so α is in $\mathcal{F}$ if and only if β is in $\mathcal{F}$. Either β contains a finite number of distinct u_i, or an infinite one.

If β contains a finite number of distinct u_i, we call them $(v_i)_{i\leq m}$. Then there is a permutation of the indexes such that the result of the permutation on β is $(v_0.v_1 \ldots v_m)^\omega$. We know that ${v_m}^\omega \in \mathcal{F}$ (see the lemma), and for all i, $\mathcal{F}(v_i) = \mathcal{F}$, so $v_0.v_1 \ldots v_{m-1}.(v_m)^\omega \in \mathcal{F}$. We call $\gamma = v_0.v_1 \ldots v_{m-1}.(v_m)^\omega$. Because $\gamma \in \mathcal{F}$, there is a sequence $(u'_i)_{i\in\mathbb{N}}$ such that $\mathcal{G}'(u'_i) = \mathcal{B}^\omega$, $\gamma = u'_0.u'_1 \ldots u'_i \ldots$ and u'_i minimum. So there is a j such that $u'_0.u'_1 \ldots u'_j = v_0.v_1 \ldots v_{m-1}.{v_m}^n.w$ with w prefix of v_m. Whatever i, $(u'_0.u'_1 \ldots u'_i)^\omega \in \mathcal{F}$, because $\mathcal{F} = \Omega\left(\mathcal{G}'\right)$. So, by Proposition 6, $(v_0.v_1 \dots v_m)^\omega \in \mathcal{F}$. This in turn means that $\beta \in \mathcal{F}$.

If β contains infinitely many distinct u_i, we call them $(v_i)_{i\in\mathbb{N}}$. Necessarily, there is infinitely many 0's and 1's in β. So we have two v_i, w_0 and w_1 such that w_0 contains a 0 and w_1 contains a 1. As β is composed of 0 and 1, there is a permutation of the entries that transforms β in $(w_0.w_1)^\omega$. So we are back to the problem with β containing a finite number of u_i.

Thus, $\beta \in \mathcal{F}$ whatever the case, which proves that $\alpha \in \mathcal{F}$. We started from $\alpha \in \Omega(\mathcal{G})$, so $\Omega(\mathcal{G}) \subset \mathcal{F}$. Because we already proved $\mathcal{F} \subset \Omega(\mathcal{G})$, we have $\mathcal{F} = \Omega(\mathcal{G})$. $\square$

It is possible to compute effctively the best open function representing an iterative function $\mathcal{F}$, provided we already have a representation, that is an open function. The idea is to detect the u such that $u^\omega \in \mathcal{F}$ and $\mathcal{F}(u) = \mathcal{F}$. How we can do this without looping is explained in [19].

7 A New Class: Regular Functions

Now that we have a wide variety of infinite behaviors, we will try to incorporate them with the finite behavior to obtain closure by boolean operations, and thus a very wide variety of infinite functions. The idea is to allow a finite set of iterative functions to describe the infinite behavior at a given point in the function.

Definition 31 *Let $\mathcal{F} : \mathcal{B}^\omega \to \mathcal{B}$ be a named function. The function $\mathcal{F}$ is said to be* regular *if and only if $\mathcal{F}$ is prefix regular and there is a finite set of non-empty iterative functions* $\mathrm{iter}(\mathcal{F})$ *such that $\forall \alpha \in \mathcal{F}$, $\exists u$, $\exists \mathcal{G} \in \mathrm{iter}(\mathcal{F})$, $\alpha \in u.\mathcal{G}$ and $\mathcal{G} \subset \mathcal{F}(u)$.*

The informal meaning of this definition is that for any vector in the relation, there is a finite point in the vector such that the tail of the vector is in one of the infinite behaviors (iterative functions) of the relation.

Example 32 *Let $\mathcal{F}$ be $\{0^\omega, 1^\omega\}$. This function is prefix regular (the different $\mathcal{F}(u)$ are just $\mathcal{F}$, $\mathcal{F}(0)$, $\mathcal{F}(1)$ and $\emptyset$). The functions $\mathcal{F}_0 = \{0^\omega\}$ and $\mathcal{F}_1 = \{1^\omega\}$ are iterative functions. Any vector in $\mathcal{F}$ is either in $\mathcal{F}_0$ or $\mathcal{F}_1$. So $\mathcal{F}$ is regular.*

Let $\mathcal{G}$ be the function `true` *on any vector ending with 0^ω or 1^ω. Whatever the vector α in $\mathcal{G}$, there is a u such that $\alpha = u.0^\omega$ or $\alpha = u.1^\omega$. In any case, $\alpha \in u.\mathcal{F}_i$ $(i = 0, 1)$. Moreover, $\mathcal{F}_i \subset \mathcal{G}(u)$. So $\mathcal{G}$ is also regular.*

These functions are called regular because of the analogy with ω-regular sets of words of Büchi [11]. The only restriction imposed by the fact that we consider functions lies in the entry names, namely the entry names must be ultimately periodic to be finitely representable.

Theorem 33 *A named function $\mathcal{F}$ is regular if and only if its entry names are ultimately periodic and the set of words in $\mathcal{F}$ is ω-regular in the sense of Büchi.*

The idea is that open functions define regular languages. If U is the regular language defined by an open function, then the associated iterative function is U^ω. The idea of the proof of the theorem is that an ω-regular language can be characterized as a finite union of $U.V^\omega$, with U and V regular languages.

PROOF. Let $\mathcal{F}$ be a regular function. Because it is prefix regular, its entry names are ultimately periodic. Let iter($\mathcal{F}$) be $(\mathcal{F}_i)_{i\in C}$. Each iterative function defines an ω-regular language: the Büchi automaton that accepts the language defined by an iterative function represented by the open function $\mathcal{G}$ is $(Q, E, \{\mathcal{G}\}, \{\mathcal{G}\})$ with:

$$\begin{aligned} Q &= \{\, \mathcal{G}(u) \mid u \in \mathcal{B}^* \} \\ E &= \{\, (\mathcal{G}(u), b, \mathcal{G}_{ub}) \mid b \in \mathcal{B},\ u \in \mathcal{B}^* \} \\ &\quad \text{where } \mathcal{G}_v = \text{if } \ (\mathcal{G}(v) = \mathcal{B}^\omega \text{ and } \mathrm{name}_{\mathcal{G}}(|v|) = \mathrm{name}_{\mathcal{G}}(0)) \ \text{ then } \mathcal{G} \\ &\quad \phantom{\text{where } \mathcal{G}_v = \text{if } \ (} \text{else } \mathcal{G}(v) \end{aligned}$$

The state set Q and the transition set E are finite because of the prefix regularity of $\mathcal{G}$.

$\mathcal{F} = \bigcup_{u\in U} \bigcup_{i\in\{\, j\in C \,\mid\, \forall\alpha\in\mathcal{F}_i,\, u\alpha\in\mathcal{F}\}} \{\, u\alpha \mid \alpha \in \mathcal{F}_i\}$ by definition of the regularity of $\mathcal{F}$. Each set $\{\, u\alpha \mid \alpha \in \mathcal{F}_i\}$ is ω-regular because ω-regularity of words is closed under concatenation, and the number of such sets is finite by prefix regularity of $\mathcal{F}$. Being a finite union of ω-regular languages, $\mathcal{F}$ represents an ω-regular language.

Now let (Q, E, I, F) be a Büchi automaton such that there is a boolean function $\mathcal{F}$ representing the same set of words and the entry names of $\mathcal{F}$ are ultimately periodic. Let $\mathcal{F}_q$ be the set of words corresponding to the Büchi automaton $(Q, E, \{q\}, \{q\})$. We define $L^*(Q, E, \{q\}, \{q\})$ to be the set of finite words represented by this automaton. Each $\mathcal{F}_q$ is an iterative function represented by the open function:

$$\mathcal{G}_q = \{\alpha \in \mathcal{B}^\omega \mid \exists u,\ \alpha = u\beta,\ u \in L^*\left(Q, E, \{q\}, \{q\}\right)\}$$

These functions are prefix regular because the automaton is finite and the entry names are ultimately periodic. They are obviously open because a finite decision procedure is enough to find out that a given infinite vector is in the relation. Whatever α in the set represented by the Büchi automaton, α is the label of an infinite path such that there is a q in F and q is in the set of infinitely repeated states of the path. Thus, there is an u such that $\alpha = u\beta$ and β is in $\mathcal{F}_q$, and for all $\gamma \in \mathcal{F}_q$, $u\gamma$ is in the language of the Büchi automaton (Q, E, I, F). So $\mathcal{F}$ is a regular function and $\text{iter}(\mathcal{F}) = (\mathcal{F}_q)_{q\in F}$. $\square$

Corollary 34 *If $\mathcal{F}$ and $\mathcal{G}$ are regular functions, then $\mathcal{F} \wedge \mathcal{G}$, $\mathcal{F} \vee \mathcal{G}$ and $\neg\mathcal{F}$ are regular functions.*

It is an immediate consequence of the theorem, the closure properties of ω-regular languages, and the closure properties of the fact that the set of entry names is ultimately periodic.

7.2 Attempting a Representation

Let us recall the definition of a regular function $\mathcal{F}$: $\forall\alpha \in \mathcal{F}$, $\exists u$, $\exists\mathcal{G} \in \text{iter}(\mathcal{F})$, $\alpha \in u.\mathcal{G}$ and $\mathcal{G} \subset \mathcal{F}(u)$. It provides a natural decision process (possibly infinite if the vector is not in the relation): for each prefix u of α (in increasing order), we test $\mathcal{F}(u)$. If is is empty, then α is not in the relation. Otherwise, for each $\mathcal{G} \in \text{iter}(\mathcal{F})$ such that $\mathcal{G} \subset \mathcal{F}(u)$, we test if the remaining of α is in $\mathcal{G}$. If one of these tests is positive, α is in the relation.

If we try to insert the necessary informations in the decision tree, we need to store at each point in the decision process the set of possible iterative functions. Representing an iterative function is easy and unique (see previous section). The first problem is that we do not have such uniqueness results for finite unions of iterative functions. The second problem is the non deterministic nature of the decision process, which consists in quite inefficient tries and backtrackings. Such a representation extending BDDs would be possible, but we would loose (as far as the author tried) too many good properties of the BDDs. So we will try in the next section to define a smaller class of infinite functions (but bigger than mere open functions) that could have a good representation.

8 BDGs with `iter` nodes: ω-deterministic Functions

In order to obtain a tractable class of functions, we restrict the class of regular functions. We call these functions ω-deterministic because we restrict the number of possible infinite behaviors at a given u to at most one iterative function.

Definition 35 *Let $\mathcal{F} : \mathcal{B}^\omega \to \mathcal{B}$ be a named function. $\mathcal{F}$ is ω-deterministic if and only if $\mathcal{F}$ is prefix regular and*

$$\forall u,\ \Omega\left(\{\, v.\alpha \mid u.v^\omega \in \mathcal{F} \text{ and } \mathcal{F}(u.v) = \mathcal{F}(u)\}\right) \subset \mathcal{F}(u)$$

The iterative function $\mathcal{F}^{\Omega}_{[u]} \stackrel{\text{def}}{=} \Omega\left(\{\, v.\alpha \mid u.v^\omega \in \mathcal{F} \text{ and } \mathcal{F}(u.v) = \mathcal{F}(u)\}\right)$ is the only infinite behavior possible at u. Any other iterative function in the infinite behavior at this point would be included into $\mathcal{F}^{\Omega}_{[u]}$.

Theorem 36 *An ω-deterministic function $\mathcal{F}$ is regular, and whatever the functions in* $\mathrm{iter}(\mathcal{F})$ *representing the infinite behavior at a given point u, there is an iterative function representing the infinite behavior at u and containing all of them.*

PROOF. Let $\mathcal{F}$ be an ω-deterministic relation. We first prove that $\mathcal{F}$ is regular. We define $\mathrm{iter}(\mathcal{F}) = \left\{ \mathcal{F}^{\Omega}_{[u]} \,\middle|\, \mathcal{F}^{\Omega}_{[u]} \neq \emptyset \right\}$. This set is finite because $\mathcal{F}$ is prefix regular. Let $\alpha \in \mathcal{F}$. If α is ultimately periodic, as $\mathcal{F}$ is prefix regular, there is an u and a v such that $\mathcal{F}(u.v) = \mathcal{F}(u)$ and $\alpha = u.v^\omega$. Then $\mathcal{F}^{\Omega}_{[u]} \neq \emptyset$, and for all $\beta \in \mathcal{F}^{\Omega}_{[u]}$, $u.\beta$ is in $\mathcal{F}$ because $\mathcal{F}$ is ω-deterministic. If α is not ultimately periodic, as the entry names are ultimately periodic, there is a permutation of the entries which transforms α in β which is ultimately periodic. Moreover, it is possible to choose the permutation that it leaves u unchanged (we start after the last letter that appears finitely many times and after the looping of the relation) such that $\beta = u.v^\omega$ as above. As the $\mathcal{F}^{\Omega}_{[u]}$ have the same equivalence of entries as $\mathcal{F}(u)$ (at least), we have the postfix of α after u is in $\mathcal{F}^{\Omega}_{[u]}$.

Concerning the canonicity of the iterative function at a given point, we have the fact that for all iterative function at u of $\mathcal{F}$, the iterative function is included in $\mathcal{F}^{\Omega}_{[u]}$. $\square$

8.1 The Decision Tree

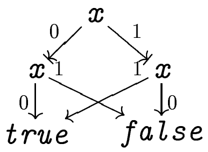
If the set $\{\, v \mid u.v^\omega \in \mathcal{F} \text{ and } \mathcal{F}(u.v) = \mathcal{F}(u)\}$ is not empty, then it is possible, in the decision process, that we enter an infinite behavior. It must be signaled in the decision tree. To this end, we introduce a new kind of node in the decision tree, the `iter` node. The `iter` node signals that we must start a new infinite behavior, because before this node, we were in fact in the finite part of the function. The `iter` node has only one child. In the graphical representation, we will sometimes write $\begin{smallmatrix} x \\ \downarrow \\ t \end{smallmatrix}$ for $\begin{smallmatrix} x \\ \downarrow \\ \texttt{iter} \\ \downarrow \\ t \end{smallmatrix}$. After a `iter` node, we start the decision tree representing the iterative function. We know that when this decision tree comes to a `true`, we must start again just after the previous `iter` node. `false` nodes in the decision tree are replaced by the sequel of the description of the ω-deterministic function. As iterative functions of the ω-deterministic function are uniquely determined, and their representation is unique, the decision tree of the ω-deterministic function is unique.

Note that open functions are ω-deterministic. Their representation as an ω-deterministic function is the same as in the previous section, but with a `iter` preceding every `true`. It means also that the restriction of this representation to finite functions give the classical BDD, except for the `iter` preceding the `true`.

8.2 The Semantics of the Decision Tree

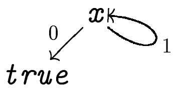
The semantics of the decision tree is defined in terms of a pseudo-decision process (it is not an actual decision process because it is infinite). The decision process reads a vector and uses a stack S and a current iterative tree, r. At the beginning, the stack is empty and r is the decision tree. When we come to a `true` node, we stack it and start again at r. When we come to a $\begin{smallmatrix} \texttt{iter} \\ \downarrow \\ t \end{smallmatrix}$ node, r becomes t and we empty the stack. If we come to a `false` node, we stop the process. The process is a success if it doesn't stop and the stack is infinite.

Example 37



when we read a 0, the current iterative tree becomes $\begin{smallmatrix} & x & \\ \swarrow & & \searrow \\ \mathit{true} & & \mathit{false} \end{smallmatrix}$. If we read a 1 after that, we stop on a failure, and if we read a 0, we stack a `true` *and start again with the same iterative tree. So, after a 0, we can only have 0^ω. After a 1, the iterative tree becomes $\begin{smallmatrix} & x & \\ \swarrow & & \searrow \\ \mathit{false} & & \mathit{true} \end{smallmatrix}$, and this time, we can only have 1^ω. So this function is $\{0^\omega, 1^\omega\}$.*

Example 38



during the decision process, the iterative tree never changes. When we read a 0, we stack a `true` *and start again. But each time we read a 1, we empty the stack. So the only vectors that stack an infinite number of* `true` *are the vectors ending by* 0^ω.

8.3 Boolean Operators

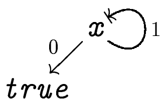
Proposition 39 *Let $\mathcal{F}$ and $\mathcal{G}g$ be two ω-deterministic functions. Then $\mathcal{F} \wedge \mathcal{G}$ is ω-deterministic.*

This property is derived from the fact that iterative functions are closed under intersection, so the infinite behavior at each point is an iterative function. Moreover, intersection preserves prefix regularity, so the result is ω-deterministic.

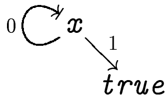
The algorithm building the decision tree representing $\mathcal{F} \wedge \mathcal{G}$ identifies the loops, that is we come from a (t, u), which are subtrees of the decision trees representing $\mathcal{F}$ and $\mathcal{G}$, and return to a (t, u). If in such a loop, we have not encountered any new `true` in any decision tree, we build a loop, if one decision process has progressed, we keep building the decision tree, and when both have been through a `true`, we add a `true` in the intersection.

ω-deterministic functions are not closed under union. As they are closed under intersection, it means that they are not closed under negation either.

Example 40 (Impossible Union) *Let $\mathcal{F}_1$ be the set of all vectors with a finite number of 1's, and $\mathcal{F}_2$ the set of all vectors with a finite number of 0's. $\mathcal{F}_1$ and $\mathcal{F}_2$ are represented by:*



and



Let $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. The set $\Omega\left(\{u.\alpha \mid u^\omega \in \mathcal{F} \text{ and } \mathcal{F}(u) = \mathcal{F}\}\right)$ is the set of all vectors, but $(01)^\omega \notin \mathcal{F}$, so $\mathcal{F}$ is not ω-deterministic.

8.4 Approximation

Proposition 41 *Whatever $\mathcal{F}$ prefix regular function, there is a least (for set inclusion) ω-deterministic function containing $\mathcal{F}$.*

The process of building the best ω-deterministic function approximating a prefix regular function consists in adding the iterative functions defined by $\{v.\alpha |\, u.v^\omega \in \mathcal{F}$ and $\mathcal{F}(u.v) = \mathcal{F}(u)\}$ to $\mathcal{F}(u)$. We define $\mathcal{F}^+ \stackrel{\text{def}}{=} \mathcal{F} \cup \bigcup_w w.\mathcal{F}^\Omega_{[w]}$. We just add the minimum number of vectors so that the infinite behaviors at each point is an iterative function. This function is also prefix regular, by prefix regularity of $\mathcal{F}$, and as such it is ω-deterministic.

If we start from an ω-deterministic function and perform operations that preserve prefix regularity, such as union, we can give best approximations of these operations. It means that we can have a kind of abstract behavior, keeping as close as possible to the desired operations, while keeping the good representation as BDGs.

9 Conclusion

To achieve the representation of infinite functions, we presented a new insight on variables which allows the sharing of some variables. This sharing is compatible with every operation on classical BDDs, at no additional cost. It is even an improvement for classical BDDs, as it speeds up one of the basic operations on BDDs, the restriction operation.

We presented three classes of infinite functions which can be represented by extensions of BDDs. So far, the only extension that allowed the representation of infinite function was presented by Gupta and Fisher in [20] to allow inductive reasoning in circuit representation. Their extension corresponds to the first class (open functions), but without the uniqueness of the representation, because the loops have a name, which is arbitrary (and so there is no guarantee that the same loop encountered twice will be shared).

Our representation for open functions and ω-deterministic function have been tested in a prototype implementation in Java. Of course, this implementation cannot compete with the most involved ones on BDDs. It is, however, one of the advantages of using an extension of BDDs: many useful optimizations developed for BDDs could be useful, such as complement edges [21] or differential BDDs [22]. This last extension could lead to wider classes of functions by releasing some of the restrictions imposed by the equivalence of entries. This is a direction for future work. Another direction for future work concerns the investigation over regular functions. These functions are closed under boolean operations, but we did not find a satisfactory unique representation with a decision tree yet. We believe the first two classes will already be quite useful. For example the first class (open function) is already an improvement over [20], and the second class (ω-deterministic) can express many useful properties of temporal logic [23]. This work is a step towards model checking and static analysis of the behavior of infinite systems, where properties depending on fairness can be expressed and manipulated efficiently using BDGs [24].

References

[1] R. E. Bryant, Graph based algorithms for boolean function manipulation, IEEE Transactions on Computers C-35 (1986) 677–691.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, J. Hwang, Symbolic modek checking: 10^{20} states and beyond, in: Fifth Annual Symposium on Logic in Computer Science, 1990.

[3] J.-C. Madre, O. Coudert, A logically complete reasoning maintenance system based on a logical constraint solver, in: 12th International Joint Conference on Artificial Intelligence, 1991, pp. 294–299.

[4] M.-M. Corsini, K. Musumbi, A. Rauzy, The μ-calculus over finite domains as an abstract semantics of Prolog, in: M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, A. Rauzy (Eds.), Workshop on Static Analysis, no. 81–82 in Bigre, 1992.

[5] B. Le Charlier, P. van Hentenryck, Groundness analysis for Prolog: Implementation and evaluation of the domain prop, in: PEPM'93, 1993.

[6] R. Bagnara, A reactive implementation of pos using ROBDDs, in: H. Kuchen, S. D. Swierstra (Eds.), 8th International Symposium on Programming Languages, Implementation, Logic and Programs, Vol. 1140 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 107–121.

[7] L. Mauborgne, Abstract interpretation using typed decision graphs, Science of Computer Programming 31 (1) (1998) 91–112.

[8] P. Cousot, R. Cousot, Abstract interpretation; a unified lattice model for static analysis of programs by construction of approximation of fixpoints, in: 4th ACM Symposium on Principles of Programming Languages (POPL '77), 1977, pp. 238–252.

[9] P. Cousot, Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes, Ph.D. thesis, Université de Grenoble (March 1978).

[10] A. Srinivasan, T. Kam, S. Malik, R. K. Brayton, Algorithms for discrete function manipulation, in: IEEE International Conference on Computer-Aided Design, 1990, pp. 92–95.

[11] J. R. Büchi, On a decision method in restricted second order arithmetics, in: E. Nagel, et al. (Eds.), International Congress on Logic, Methodology and Philosophy of Science, Stanford University Press, 1960.

[12] A. P. Sistla, M. Y. Vardi, P. Wolper, The complementation problem for büchi automata, with applications to temporal logic, in: 12th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Springer-Verlag, 1985.

[13] M. Fujita, H. Fujisawa, N. Kawato, Evaluation and improvements of boolean comparison method based on binary decision diagram, in: International Conference on Computer-Aided Design, IEEE, 1988, pp. 3–5.

[14] S.-W. Jeong, B. Plessier, G. D. Hachtel, F. Somenzi, Variable ordering and selection for FSM traversal, in: International Conference on Computer-Aided Design, IEEE, 1991, pp. 476–479.

[15] L. Mauborgne, Improving the representation of infinite trees to deal with sets of trees, in: G. Smolka (Ed.), European Symposium on Programming (ESOP 2000), Vol. 1782 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 275–289.

[16] L. Mauborgne, An incremental unique representation for regular trees, Nordic Journal of Computing 7 (4) (2000) 290–311.

[17] R. E. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Computing Surveys 24 (3) (1992) 293–318.

[18] D. Michie, "memo" functions and machine learning, Nature 218 (1968) 19–22.

[19] L. Mauborgne, Representation of sets of trees for abstract interpretation, Ph.D. thesis, École Polytechnique (1999).

[20] A. Gupta, A. L. Fisher, Parametric circuit representation using inductive boolean functions, in: C. Courcoubetis (Ed.), Computer Aided Verification, Vol. 697 of Lecture Notes in Computer Science, Springer-Verlag, 1993, pp. 15–28.

[21] J. P. Billon, Perfect normal form for discrete programs, Tech. Rep. 87039, BULL (1987).

[22] A. Anuchitanakul, Z. Manna, T. E. Uribe, Differential BDDs, in: J. van Leeuwen (Ed.), Computer Science Today, Vol. 1000 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 218–233.

[23] Z. Manna, A. Pnueli, The anchored version of the temporal framework, in: J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, Vol. 354 of Lecture Notes in Computer Science, Springer-Verlag, 1988, pp. 201–284.

[24] L. Mauborgne, Tree schemata and fair termination, in: J. Palsberg (Ed.), Static Analyis Symposium (SAS'00), Vol. 1824 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 302–320.