Theories and Applications of Boolean Algebras

Ohad Asor

Work-in-progress draft version 0.1, 12 March, 2024

Contents

CHAPTER 1

Preface

This monograph presents methods and results related to the first order theory of Boolean algebras and extensions thereof, which I developed during my [ongoing] work in IDNI AG designing and developing the Tau product family, in particular the Tau language which is a combination of logics described here: NSO, GSSOTC, and extensions to the first order theory of Boolean Algebras. I hope the reader will get the impression that the theories of Boolean algebras are immensely useful, and that difficult questions may become exceptionally easy using Boolean-algebraic tools, due to the unique well-behaveness of those algebras, and in particular the atomless ones.

The main four contributions of this monograph are 1. the language NSO, 2. the language GSSOTC, 3. decidable conservative extensions to the first order theory of Boolean Algebras, and 4. related algorithms. The first solves a long-lasting problem of finding a logic than can consistently refer to its own sentences. The second is a novel temporal logic. We further show applications to Description Logic and the two-variable fragment of first order logic.

The methods here are protected from commercial use by being patented.

It is my wish that mathematicians, computer scientists, and especially logicians, will find this field as fascinating as I find it, and in fact as the older generations of logicians found them (maybe most notably Alfred Tarski). Boolean algebras were indeed studied extensively in the past, but unfortunately much less the field of Boolean Functions and Equations (over general Boolean algebras), the latter four words being the title of a book by Rudeanu which is apparently the best source to this subject. It has been over 50 years since that book was published and since then very little literature touched the subject. It so happens that Boolean functions and equations are strongly connected to the first-order theory of Boolean algebras and their decision procedures. The connection between Boolean algebras and logic need not even be mentioned as it is so obvious to anyone in those and related fields. The field of Algebraic Logic contributed much to the formalization of logic in algebraic means (and in particular Boolean-algebraic means), yet, to my knowledge, yielded very little algorithmic results. I hope that readers that find the topics in this text interesting will continue researching for new logical languages and algorithms incorporating the outstanding computational and mathematical properties of Boolean algebras, Boolean function, and equations.

Finally, I would like to thank Enrico Franconi and Pawel Parys for helping me in this work.

CHAPTER 2

Preliminaries and Prior Art

This chapter introduces some basic facts about Boolean Algebras and Boolean Functions. Our notation mixes set-theoretic notation with the ring-theoretic notation. This will prove to be very useful. Contrary to the rest of this monograph, everything in this chapter is prior art.

2.1. Boolean Algebras and Boolean Functions

DEFINITION 2.1. A *Boolean Ring* (BR) is a unital ring satisfying $xx = x$ for all x in the ring.

Instead of the usual axiomatization of Boolean algebras (and its dozens of equivalent variations), we define Boolean algebras by relying on the definition of rings. Similarly we approach Boolean functions[1] by relying on the notion of polynomials over a ring.

DEFINITION 2.2. A *Boolean Algebra* (BA) is a sextuple $(B, \cap, \cup, ', 0, 1)$ where B is a BR, $\wedge$ (conjunction) is just the ring multiplication, $\vee$ (disjunction) is defined by $x \cap y = x + y + xy$, and $'$ (complementation) is defined by $x' = 1 + x$. $0, 1$ are the ring's $0, 1$.

We defined BAs using BRs but it is also possible to go the other way around by defining $x + y = xy' \vee x'y$. All BRs are therefore BAs and vice versa. We will therefore mix the notations and allow expressions like $(a + b'c) \cup d$.

The Boolean algebraic operators are usually denoted by $\wedge, \vee, \neg$ but we shall reserve those symbols to denote logical connectives. So the BA's meet and join will be denoted by juxtaposition or $\cdot$ or $\cap$, and $\cup$, respectively, complementation using $'$, and symmetric difference (ring sum) by $+$[2].

DEFINITION 2.3. A Boolean Function (BF) of n variables over a BA $\mathcal{B}$ is a multivariate polynomial function $\mathcal{B}^n \to \mathcal{B}$.

[1]Many authors define the term Boolean Function in fundamentally different ways. Similarly, and unfortunately, some authors confuse "Boolean" with "Binary".

[2]The symbol $+$ is again not to be confused with notation by other authors using it for union (or disjunction or join).

Note that we distinguish between polynomials and polynomial functions. A polynomial is a formal object that may contain arbitrary powers, however when treated as a function, idempotency comes into play and all powers are eliminated since $x^n = x$. We will always consider polynomial functions rather polynomials. Hence whenever we say "polynomial", we merely use it as a shorthand to "polynomial function".

By a monomial we shall refer to product of variables and a single constant (the "coefficient").

Definition 2.4. A Simple Boolean Function (SBF) of n variables is a BF that can be written s.t. the monomials' coefficients are either 0 or 1.

In other words, an SBF is a Boolean combination of variables, while a BF is a Boolean combination of variables and constants. Yet another way to state it, is that an SBF returns either 0 or 1 for all 2^n possible substitutions of $0, 1$. BFs and SBFs form a BA in their own right w.r.t. pointwise operations.

Theorem 2.1. *Any BR is a commutative ring. Further, for any x in a BR we have $x + x = 0$.*

Proof. By definition, $(x + x)^2 = x + x$. Expanding, we get $x + x + x + x = x + x$ implying $x + x = 0$. For commutativity, write $(x + y)^2 = x + y$. This expands and simplfies into $xy = -yx$ which is same as $xy = yx$ since we have just shown that $x = -x$. $\square$

Definition 2.5. In a BA, define a partial order $\leq$ by $x \leq y$ iff $xy' = 0$.

It is easy to verify that this is a partial order indeed in which no element is below 0 nor above 1. It is also a lattice, precisely complemented distributive lattice[3]. We will also write $x < y$ for the case that $x \leq y$ and $x \neq y$.

The following proposition can trivially be verified:

Proposition 2.1. *In any BA, the following holds for all x, y, z in the BA:*

(1) $\cap, \cup$ are commutative and associative
(2) $x(x \cup y) = x \cup xy = x$
(3) $x \cup yz = (x \cup y)(x \cup z)$
(4) $x(y \cup z) = xy \cup xz$

[3]It is indeed an interesting feature of BAs that they are an algebraic object, as well as an order theoretic object, and even logical and topological objects as well known.

(5) $x \cup 1 = x$
(6) $xx' = x + x = 0$
(7) $x'' = x$
(8) $(xy)' = x' \cup y'$
(9) $(x \cup y)' = x'y'$
(10) $x \cup y = 0$ iff $x = y = 0$
(11) $xy = 1$ iff $x = y = 1$
(12) $x \leq y$ iff $x \cup y = y$ iff $xy = x$
(13) $x \leq y$ iff $y' \leq x'$

Definition 2.6. The *two-element BA* is the BA containing only the elements $0, 1$.

Clearly it is also the finite field $\mathbb{F}_2$.

Definition 2.7. A *minterm* of n variables, denoted by X^A, is a product $x_1^{a_1}x_2^{a_2}\dots x_n^{a_n}$ where $A \in \{0,1\}^n$ and $x_i^1 = x_i; x_i^0 = x_i'$.

If $xy = 0$ then we say that x, y are *disjoint*. So 0 is disjoint from all elements including itself. The following proposition is trivial:

Proposition 2.2. *Two minterms over n variables are disjoint iff they are not equal.*

Definition 2.8. A function $\mathcal{B}^n \to \mathcal{B}$ is in *minterm normal form* if it can be written as

$$f(X) = \bigcup_{A \in \{0,1\}^n} c_A X^A$$

where $c_A \in \mathcal{B}$.

Clearly any function in minterm normal form is a BF. The converse is also true. cf. [**rud1**] for the proof of the following theorem:

Theorem 2.2. *A function $\mathcal{B}^n \to \mathcal{B}$ is a BF iff it can be written in minterm normal form*

$$f(X) = \bigcup_{A \in \{0,1\}^n} f(A)\, X^A$$

Note that we use X, A as tuples of variables, so the notation $f(X), f(A)$ should be clear.

Corollary 2.1. *A BF is uniquely determined by its values over the two-element BA.*

The following is immediate:

PROPOSITION 2.3. *Any univariate BF can be written uniquely in the form $f(x) = ax + b$ as well as the forms $f(x) = ax + bx', f(x) = ax \cup bx'$.*

Observe that if $ab = 0$ then $a + b = a \cup b$. This is sometimes useful. In particular, the latter two representations of f are using the same a, b. Also note that $a = f(1); b = f(0)$ so $f(x) = xf(1) + x'f(0)$.

The following is called Boole's normal form (sometimes mistakingly called Shannon's normal form), which, over the two-element BA, captures the ternary operation of if-then-else:

COROLLARY 2.2. *Any Boolean $f : B^n \to B$ can be written uniquely in the form $f(x_1, \ldots, x_n) = x_1 g(x_2, \ldots, x_n) + x_1' h(x_2, \ldots, x_n)$ where the unique g, h are $g(x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n)$ and $h(x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n)$.*

Another way to write BFs is called here Conjunctive Boole's normal form:

LEMMA 2.1. *Any Boolean $f : B^n \to B$ can be written uniquely in the form $f(x_1, \ldots, x_n) = (x_1' \cup g(x_2, \ldots, x_n))(x_1 \cup h(x_2, \ldots, x_n))$ where the unique g, h are $g(x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n)$ and $h(x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n)$.*

Boolean combination of functions in Boole's normal form is very easy. We show this result in its generality:

LEMMA 2.2. *Let $f(x) = ax + bx', g(x) = cx + dx'$ be unary Boolean functions and $h(x, y)$ a binary Boolean function. Then $h(f(x), g(x)) = h(a, c)x + h(b, d)x'$.*

PROOF. Applying corollary 2.2:

$$h(f(x), g(x)) = xh(f(1), g(1)) + x'h(f(0), g(0)) = xh(a, c) + x'h(b, d)$$

□

A famous result is Stone's representation theorem for Boolean algebras:

THEOREM 2.3 (Stone's Representation Theorem). *Any BA is isomorphic to a BA of sets, where $\cap$ is the usual set intersection, $\cup$ is set union, $'$ is set complement, and $\leq$ is set containment.*

This clearly justifies our notation. We will not prove this theorem here. We will only mention that each BA element is identified with a set containing all ultrafilters that contain that element. An ultrafilter is nothing but a ring homomorphism from the BR into the two-element

BA. An element belongs to the ultrafilter if the homomorphism sends it to 1.

Stone went further and showed that this set representation of BAs is in fact a topological one. We shall not deal with it here.

Indeed by homomorphism we always mean ring homomorphism which coincides with the intuitive notion of BA homomorphism.

DEFINITION 2.9. An nonzero element x in a BA is an *atom* if does not exists y s.t. $0 < y < x$.

DEFINITION 2.10. A BA is *atomless* if it contains no atoms.

DEFINITION 2.11. A BA is *atomic* if for each nonzero x in it, there exists an atom y s.t. $y \le x$.

Note that a BA may be neither atomic nor atomless. The following proposition is immediate, cf. [**kop**]:

PROPOSITION 2.4. *An nonzero element x in a BA is an atom if for all y in the BA, either $x \le y$ or $x \le y'$.*

The following result is well-known and follows from a back-and-forth argument, but we shall omit the proof here:

THEOREM 2.4. *All countable atomless BAs are isomorphic.*

Further, Tarski gave conditions under which two Boolean algebras are elementarily equivalent (meaning that any sentence in the first order theory of BA is true in one iff it's true in the other). We will not need this full result here. We will only mention that:

THEOREM 2.5. *All atomless BAs are elementarily equivalent, and all infinite atomic BAs are elementarily equivalent.*

THEOREM 2.6. *For any BF f we have*

$$f(x)\, f(x') = f(0)\, f(1)$$
$$f(x) \cup f(x') = f(0) \cup f(1)$$
$$f(x) + f(x') = f(0) + f(1)$$

PROOF. Write f in the form $f(x) = ax + bx'$. Then by applying proposition 2.2:

$$\begin{gathered} f(x)\, f(x') = (ax + bx')(bx + ax') \\ = abx + abx' = ab = f(0)\, f(1) \\ f(x) \cup f(x') = (ax + bx') \cup (bx + ax') \\ = (a \cup b)\, x + (a \cup b)\, x' = a \cup b = f(0) \cup f(1) \end{gathered}$$

$$f(x) + f(x') = (ax + bx') + (bx + ax')$$
$$= (a + b)x + (a + b)x' = a + b = f(0) + f(1)$$

□

We now observe the following property of BFs which may be seen as a first demonstration of their outstanding convenient properties:

COROLLARY 2.3. *Let f be a BF over a BA $\mathcal{B}$, then*

$$\bigcap_{x \in \mathcal{B}} f(x) = f(0)\, f(1)$$

$$\bigcup_{x \in \mathcal{B}} f(x) = f(0) \cup f(1)$$

PROOF. Direct application of theorem 2.6. □

2.2. Boolean Equations

DEFINITION 2.12. A *system of Boolean equations* of n variables and k equations is a system of the form $\{f_i(X) = 0\}_{i=1}^{k}$ where $X \in \mathcal{B}^n$ and each f_i is a BF.

Observe that an equation of the form $f(x) = g(x)$ can be written as $f(x) + g(x) = 0$, and an equation of the form $f(x) \leq g(x)$ can be written as $f(x)\, g'(x) = 0$. Therefore the definition of a system of equations covers also arbitrary equality and order constraints.

PROPOSITION 2.5. *Any system of Boolean equations is equivalent to a single equation.*

PROOF. $\{f_i(X) = 0\}_{i=1}^{k}$ holds iff $\bigcup_{i=1}^{k} f_i(X) = 0$ does. □

We follow the terminology in [**rud1, rud2**]:

DEFINITION 2.13. A *Generalized System of Boolean Equations* (GSBE) is either a Boolean equation, or the negation of a Boolean equation (namely involving $\neq 0$), or a finite combination of GSBEs by means of logical conjunction and disjunction.

REMARK 2.1. We will sometimes write systems of equations and inequations in the form of GSBEs but we will not specify explicitly that the system is finite. It should be clear that the system *always* contains finitely many equations and inequations unless specified otherwise.

DEFINITION 2.14. An *Elementary GSBE* is a system of the form $f(X) = 0, g_1(X) \neq 0, \ldots, g_k(X) \neq 0$.

Note that we cannot squash many inequations into one in the same fashion as proposition 2.5.

Now we describe Boole's consistency condition. A system of equations (or a GSBE) is *consistent* if it has a solution. We have already seen that a system of Boolean equations can be written as a single equation of the form $f(X) = 0$. The following was discovered by Boole:

THEOREM 2.7. *Let $f : \mathcal{B}^n \to \mathcal{B}$ be a BF, then $\exists X.f(X) = 0$ iff*

$$\bigcap_{A\in\{0,1\}^n} f(A) = 0$$

and $\exists X.[f(X) \neq 0]$ iff

$$\bigcup_{A\in\{0,1\}^n} f(A) \neq 0$$

PROOF. For the first statement cf. [**rud1, bro**]. The second statement is trivial. □

Note that this theorem is actually a case of quantifier elimination.

Another very important result is the Lowenheim's General Reproductive Solution (LGRS):

THEOREM 2.8. *Let $f : B^n \to B$ be a BF, and assume $f(Z) = 0$ for some $Z \in B^n$. Then the set $\{X \in B^n | (X) = 0\}$ equals precisely the image of $\phi : B^n \to B^n$ defined by $\phi(X) = Zf(X) + Xf'(X)$. Decyphering the abuse of notation, this reads $\phi_i(X) = z_i f(X) + x_i f'(X)$.*

cf. [**rud1, bro**] for a proof. One of the morals of this theorem may be stated as "if you know one solution, then you know all solution".

REMARK 2.2. The R in LGRS which stands for reproductive, means that $\forall X.f(X) = 0 \leftrightarrow \phi(X) = X$. In particular, $\forall X.\phi(\phi(X)) = \phi(X)$.

Two more important facts in which we'll make use of are:

THEOREM 2.9. *Let $f : \mathcal{B} \to \mathcal{B}$ be a BF s.t. $f(0)f(1) = 0$, or equivalently, $\exists x.f(x) = 0$. Then f(x) = 0 iff $x = t + f(t)$ for some t, iff $f(0) \leq x \leq f'(1)$.*

PROOF. For the first equivalence, write $f(x) = ax + b$. Then

$$f(x + f(x)) = a(x + ax + b) + b = ax + ax + ab + b = ab + b = f(0)f(1) = 0$$

and for the other direction, if $f(x) = 0$ then just put $t = x$. For the second equivalence, write $f(x) = ax \vee bx'$, then

$$f(x) = 0 \leftrightarrow (ax = 0) \wedge (bx' = 0) \leftrightarrow (x \leq a') \wedge (b \leq x)$$

□

We now describe the method of successive elimination (cf. [**rud1, rud2, bro**]):

THEOREM 2.10. *Let $f:\mathcal{B}^n\to\mathcal{B}$ be a BF. Set $f_n=f$ and*

$$f_k\left(x_1,\dots,x_k\right)=f_{k+1}\left(x_1,\dots x_k,0\right)f_{k+1}\left(x_1,\dots x_k,1\right)$$

Then $X\in\mathcal{B}^n$ satisfies $f\left(X\right)=0$ iff $\bigcap_{A\in\{0,1\}^n}f\left(A\right)=0$ (namely the equation satisfies the consistency condition) and

$$f_k\left(x_k=0\right)\le x_k\le\left[f_k\left(x_k=1\right)\right]'$$

For clarity, let us write it explicitly for $n=3$. The consistency condition reads

$$f\left(0,0,0\right)f\left(0,0,1\right)f\left(0,1,0\right)f\left(0,1,1\right)f\left(1,0,0\right)f\left(1,0,1\right)f\left(1,1,0\right)f\left(1,1,1\right)=0$$

and if it holds, all solutions are described by

$$f_1\left(0\right)\le x_1\le f_1'\left(1\right)$$
$$f_2\left(x_1,0\right)\le x_2\le f_2'\left(x_1,1\right)$$
$$f_3\left(x_1,x_2,0\right)\le x_3\le f_3'\left(x_1,x_2,1\right)$$

where

$$f_1\left(x_1\right)=f_2\left(x_1,0\right)f_2\left(x_1,1\right)=f\left(x_1,0,0\right)f\left(x_1,0,1\right)f\left(x_1,1,0\right)f\left(x_1,1,1\right)$$
$$f_2\left(x_1,x_2\right)=f_3\left(x_1,x_2,0\right)f\left(x_1,x_2,1\right)=f\left(x_1,x_2,0\right)f\left(x_1,x_2,1\right)$$
$$f_3\left(x_1,x_2,x_3\right)=f\left(x_1,x_2,x_3\right)$$

We can also write f_k in explicit, non-recursive form

$$f_k\left(x_1,\dots,x_k\right)=\bigcap_{a_1\in\{0,1\}}\bigcap_{a_2\in\{0,1\}}\cdots\bigcap_{a_{n-k}\in\{0,1\}}f\left(x_1,\dots x_k,a_1,\dots a_{n-k}\right)$$

2.3. The Theory of Boolean Algebra

2.3.1. General Form. The [first-order] theory of BA is simply a first-order axiomatization of the Boolean operations. Formulas in the language of BR can be may be described by the grammar

$$\phi:=atom|\neg\phi|\phi\wedge\phi|\exists var.\phi$$
$$atom:=bf=0$$
$$bf:=var|const|bf+bf|bf\cdot bf$$

Virtually all authors consider only the constants 0,1. However this clearly makes each BF in the grammar merely an SBF. We give strong focus to theories of BA interpreted in some fixed BA, enhanced with constants to each BA element interpreted by their corresponding element indeed. This allows us broader abilities to model-check fixed

BAs. To my knowledge, such theories (containing the above constants) and in particular their decidability properties and algorithms, were not studied as such, but in the form of GSBEs as above, and even then, very little literature is available. The main focus of this monograph is to investigate and extend such theories.

Given a formula, we can write it in prenex normal form or negated prenex normal form, such that the innermost quantifier is existential. Further we can write the matrix in DNF and push the innermost existential quantifier under the disjunctions. This way we can focus on studying an existential quantifier followed by an elementary GSBE.

2.3.2. Minterm Normal Form. Observe that

$$a \cup b = 0 \leftrightarrow a = 0 \wedge b = 0$$

and recall that each BF can be written as a sum of minterms, or in DNF (note that writing a BF in DNF is not the same thing as writing a formula in DNF). This allows an alternative syntax for theories of BA in which atomic formulas are of the form $cX^A = 0$. We refer to this form *minterm normal form*. Note that this is not the same minterm normal form of BFs, as here it applies to formulas.

REMARK 2.3. This normal form puts a bound on the number of quantifier-free logically equivalent formulas with n free variables and k constants. The accounting is as follows: the formula is itself an SBF of atomic formulas, and there are 2^{2^N} different SBFs in N variables. In our case N is the number of possible minterms which is readily $k2^n$. We therefore end up with a triple exponential $2^{2^{k2^n}}$ upper bound.

2.3.3. Order Normal Form. We present another normal form that might be useful in certain cases. It depends on choice of one variable. Typically the chosen variable is the one in the innermost quantifier, while the subformula following it is indeed quantifier-free.

In order normal form wrt x, each atomic formula is of the form $a \leq x \leq b$. In DNF, similar to the general normal form and in contrast to minterm normal form, it is possible to have only one positive atomic formula in each DNF clause. Another way to write a DNF clause is order normal form is:

$$\begin{gathered} a \leq x \leq b \\ \{x \nleq c_i\}_{i \in I} \\ \{d_j \nleq x\}_{j \in J} \end{gathered}$$

It is easy to see how to convert the general form to order normal form. Positive atoms

$$f(x, X) = 0$$

are

$$f(0,X) \leq x \leq f'(1,X)$$

and negative atoms

$$f(x,X) \neq 0$$

are

$$[f(0,X) \nleq x] \vee [x \nleq f'(1,X)]$$

2.3.4. Complexity. Quantifier elimination in theories of BA where constants are either 0,1 were studied by Tarski by introducing the so-called invariants. Kozen extended this notion of invariants and by that derived the specific complexity for decision procedure. For infinite BAs, it is complete for $\bigcup_c \mathrm{STA}(*, 2^{cn}, n)$. Roughly, this means anything that can be done in exponential time by an alternating Turing machine with linearly many alternations. For the two-element BA, it is simply QBF which is maybe the most famous PSPACE-complete problem.

2.4. Lindenbaum-Tarski Algebras

Lindenbaum-Tarski Algebras (LTAs) are obtained by taking any logic in which its formulas are closed under conjunction, disjunction, and negation, quotiened by logical equivalence. Therefore they form a BA. This BA may be atomic or atomless or neither: looking at a formula as a set of models (which is justified because formulas are considered only up to logical equivalence), then a formula that has a single model, is clearly an atom in the LTA. Observe that every can be seen as an ultrafilter.

Remark 2.4. An important example of an atomless LTA is for a logic in which its signature is infinite. This observation is trivial and is left to the reader. Also observe that, if looking at BA elements as sets (as justified by Stone's theorem), whether or not set of models as in LTA, then every element in any atomless BA, except 0, is an infinite set.

2.5. Hall's Marriage Theorem

Our treatment of GSBEs will involve Hall's marriage theorem. We present it here in its set-theoretical version:

Definition 2.15. Let $A_1, \ldots, A_n$ be sets, not necessarily distinct. A choice of elements $a_1 \in A_1, \ldots, a_n \in A_n$ such that $a_i \neq a_j$ for all $i \neq j$ is called a *system of distinct representatives.*

THEOREM 2.11. *Let $\mathcal{A} = A_1, \ldots, A_n$ be a sequence of sets, not necessarily distinct. Then $\mathcal{A}$ does not have a system of distinct representatives, iff there exists a subsequence $\mathcal{B} = B_1, \ldots, B_m$ of $\mathcal{A}$ s.t. $|\bigcup_i B_i| < m$.*

This condition of nonexistence is commonly translated to nonexistence of an X-saturated matching in a bipartite graph, and efficient algorithms exist for this decision problem. Finding the subsequence $\mathcal{B}$ (if exists) is commonly referred to as finding a "Hall Violator".

Remarkably, the consistency of GSBEs comes down directly to Hall's theorem, to be demonstrated here later on, which is something that apparently all authors dealing with GSBEs have overlooked.

A simple observation which we shall make use of later on is that a system of distinct representative exists iff it exists for the subsequence in which all infinite A's are removed from it. In other words, infinite sets in a family of sets don't influence the existence of distinct representative.

CHAPTER 3

Quantifier Elimination

In this chapter we shall present methods for deciding formulas in the language of BA by means of quantifier elimination.

3.1. Distinct Representatives

Several results have been published regarding quantifier elimination in BAs, going back to Tarski [], and continuing through [,,,,] to mention only a few examples. However some of them do not offer any convenient or [relatively] efficient algorithm, especially not when compared to our algorithm, and moreover some of the statements in the literature (concerning either quantifer elimination or GSBEs) even have easy counterexamples. Many of the relevant proofs in the literature are also very hard to verify. Furthermore, some results were not accompanied with proofs or algorithms, but only with examples which do not seem to demonstrate the general case nor their correctness even on special cases. Here we shall give simple, elementary statements, proofs, and algorithms, for consistency conditions of GSBEs (and in turn for quantifier elimination in theories of BAs) in a fashion that completely settles this topic. The atomless case is easy, both conceptually and algorithmically, and offers a full quantifer elimination method. The non-atomless case is much more demanding, and offers quantifier elimination only into theories strictly richer than theories of BA. We begin with the general case and then point out the differences between atomless and non-atomless algebras.

Theorem 3.1. *Let $X^{A_1},\dots,X^{A_m}$ be minterms in n variables, and $b_1,\dots,b_m$ elements in some BA. Then*

$$\exists X.\bigwedge_{i=1}^{m} X^{A_i} \geq b_i$$

iff $b_ib_j = 0$ whenever $A_i \neq A_j$.

Proof. First assume that $X^{A_1},\dots.X^{A_m}$ are all distinct and therefore the nonzero b's are all disjoint, otherwise convert any two equations of the form

$$X^{A_i} \geq s$$
$$X^{A_i} \geq t$$

into the equivalent form $X^{A_i} \geq s \vee t$. Necessity is now immediate recalling that two different minterms are always disjoint and that subsets of disjoint sets must also be disjoint. For sufficiency and $n = 1$ the equations take the form $x \geq b_1$ and $x' \geq b_2$ which indeed holds iff $b_1 b_2 = 0$. Assume for n and consider an additional variable x. Then we can split the equations into $p + q = m$ equations and rewrite them as

$$\left\{xX^{A_i} \geq b_i\right\}_{i=1}^{p}$$
$$\left\{x'X^{B_j} \geq c_j\right\}_{j=1}^{q}$$

and let X be a solution of

$$\left\{X^{A_i} \geq b_i\right\}_{i=1}^{p}$$
$$\left\{X^{B_j} \geq c_j\right\}_{j=1}^{q}$$

by the induction hypothesis after making sure that all A_i, B_i are disjoint (while if $p + q = 1$ then a solution trivially exists). If $p \neq 0$, set $x = \bigcup_k b_k$. Then $\bigcup_k c_k \leq x'$ due to the disjointness assumption. Therefore

$$xX^{A_i} = \left(\bigcup_k b_k\right) \wedge X^{A_i} \geq b_i X^{A_i} = b_i$$

$$x'X^{B_j} \geq \left(\bigcup_k c_k\right) X^{B_j} \geq c_j X^{B_j} = c_j$$

Similarly set $x = \bigcap_k c'_k$ if $p = 0$, or simply $x = 0$. □

Corollary 3.1. *The system* $\left\{b_i X^{A_i} \neq 0\right\}_{i=1}^{m}$ *has a solution iff there exists* $0 < c_i \leq b_i$ *s.t.* $c_i c_j = 0$ *whenever* $A_i \neq A_j$.

The condition in the corollary is completely equivalent to theorem 2.11 once treating each b_i as follows: if it can be written as a disjunction of atoms, then we treat it as a set whose elements are those atoms, and each c_i is a choice of an atom. If b_i cannot be written as a union of atoms, then we treat it as an infinite set and by that it is eliminated from the problem as we have pointed out after theorem 2.11.

This gives a complete characterization and algorithm for quantifier-elimination in theories of fixed BAs (namely theories where the BA is given in contrast to theories concerning all or several BAs), by eliminating all equalities by first using proposition 2.5 and then using the LGRS, a procedure which in turn eliminates all equality constraints, then writing all inequations in minterm normal form, and asking whether a combination of minterms exists (one from each inequation) s.t. no Hall violator exists.

We now make the quantifer elimination explicit. In the non-atomless case, the quantifier is eliminated into a statement in a richer language (e.g. language with cardinalities), to a condition saying that no Hall violator exists, or that a corresponding bipartite matching exists. In the atomless case, first observe that a system of the form $g_1(X) \neq 0, \dots, g_n(X) \neq 0$ is consistent iff none of the g's is identically zero. Write it in the form $g_1(x, X) \neq 0, \dots, g_n(x, X) \neq 0$ and we'd like to express the same condition such that x is eliminated. This is readily done by writing $g_1(0, X) \cup g_1(1, X) \neq 0, \dots, g_n(0, X) \cup g_n(1, X) \neq 0$ due to corollary 2.1. We formulate one of those observations in a corollary to be used later on:

COROLLARY 3.2. *Multivariate BFs over an atomless BA have a common nonzero iff none of them is identically zero.*

THEOREM 3.2. *In atomless BA, the system*

$$f(x) = 0 \wedge \bigwedge_{i \in I} g_i(x) \neq 0$$

has a solution iff

$$f(0) f(1) = 0 \wedge \bigwedge_{i \in I} g_i(f(0)) \cup g_i(f'(1)) \neq 0$$

has a solution.

PROOF. Using the last corollary and theorem 2.9. □

3.2. The Atomic Case

Fix an atomic Boolean algebra $\mathcal{B}$.

THEOREM 3.3. *The system* $\{a_i x \neq 0\}_{i=1}^{N}, \{b_j x' \neq 0\}_{j=1}^{K}$ *has a solution iff there exist atoms* s_i, t_j *s.t.*

$$\{s_i \leq a_i\}_{i=1}^{N}, \{t_j \leq b_j\}_{j=1}^{K}, \forall ij . s_i \neq t_j$$

in which case any $\bigcup_i s_i \leq x \leq \bigcap_j t'_j$ *is a solution.*

PROOF. Verifying $\bigcup_i s_i \leq x \leq \bigcap_j t'_j$ as solutions is straightforward. For the other way around, if x satisfies $\{a_i x \geq c_i\}_{i=1}^{N}, \{b_j x' \geq d_j\}_{j=1}^{K}$ for some nonzero c_i, d_j, then any choice of atoms from c_i, d_j will satisfy our requirements as c_i must be disjoint from any d_j. □

COROLLARY 3.3. *The system* $\{a_i x \neq 0\}_{i=1}^{N}, \{b_j x' \neq 0\}_{j=1}^{K}$ *has a solution iff it has a solution of cardinality at most* N.

PROOF. This is because $x = \bigcup_{i=1}^{N} s_i$ is a solution, as above. $\square$

COROLLARY 3.4. *A formula in the language of BA containing n variables interpreted over $\mathcal{B}$ is true iff it's true in an algebra of size $2^{2^{n-1}}$.*

PROOF. We relativize quantifiers successively as follows. Without loss of generality we deal only with existentially quantified single DNF clause of the form

$$\exists x.f\left(x\right) = 0 \wedge \bigwedge_{i} g_i\left(x\right) \neq 0$$

which can be written as:

$$\left[f\left(0\right) f\left(1\right) = 0\right] \wedge \exists x. \bigwedge_{i} g_i\left(x + f\left(x\right)\right) \neq 0$$

and can be converted into the form:

$$\left[f\left(0\right) f\left(1\right) = 0\right] \wedge \exists x. \bigwedge_{i} a_i x \neq 0 \wedge \bigwedge_{i} b_i x' \neq 0$$

where a_i, b_i are minterms in the remaining variables. Since there are no more than 2^{n-1} minterms in the n variables excluding x, this formula can be relativized as:

$$\left[f\left(0\right) f\left(1\right) = 0\right] \wedge \exists \left|x\right| \leq 2^{n-1}. \bigwedge_{i} a_i x \neq 0 \wedge \bigwedge_{i} b_i x' \neq 0$$

$\square$

We shall see more forms of quantifier elimination in the next section.

CHAPTER 4

Finding Solutions

4.1. In General and Minterm Normal Form

First we present a way to find a single zero of a BF, which in turn allows to then characterize all zeros by LGRS. The following theorem should be understood recursively, so we find a substitution for each variable and move on to the next variables.

THEOREM 4.1. *For $f(x,X) = xg(X) + x'h(X)$, let Z be a zero of $g(Z)h(Z)$ (which is guaranteed to exist by Boole's consistency condition). Then both $f(h(Z),Z) = 0$ and $f(g'(Z),Z) = 0$.*

PROOF. Exercise. $\square$

We now deal with finding solutions for elementary GSBE in atomless BA. Consider

$$f(X) = 0$$

$$\{g_i(X) \neq 0\}_{i\in I}$$

and let ϕ be the LGRS of f (wrt some arbitrarily chosen single zero of f), and assume that a solution to the whole system, exists. Set $h_i(X) = g_i(\phi(X))$ and suppose T satisfies $\{h_i(T) \neq 0\}_{i\in I}$, then $f(T) = 0$ because the LGRS is reproductive (cf. remark 2.2). So to solve the original system we only need to solve $\{h_i(T) \neq 0\}_{i\in I}$ and the solution to the original system is then $\phi(T)$. To this end, for each h_i we find a bitstring H_i s.t. $h_i(H_i) \neq 0$. This is the same as writing h_i in minterm normal form (alternatively DNF), choosing one minterm (which corresponds to H_i), and $h_i(H_i)$ will yield the coefficient of that minterm. We now get a system of the form

$$X^{H_i} h_i(H_i) \neq 0$$

(the "minterm system" hereby) which clearly depends on the choice of H_i but any such single choice, if has a solution, will yield a solution to the original system, and vice versa: if a solution to the original system exists, then such a choice exists.

For runtime optimization considerations, there are two things to bear in mind here:

1. The more disjoint the $h_i\,(H_i)$'s are, namely $h_i\,(H_i)\,h_j\,(H_j)=0$, the less effort we'll need to invest in order to make the minterm system disjoint (cf. the next exercise).

2. The more $H_i = H_j$, the less minterms will be involved in the final system.

Solving the minterm system can be done by:

THEOREM 4.2. *The system*

$$\begin{aligned}&\left\{xX^{A_i}=0\right\}_{i\in I_1}\\&\left\{x'X^{B_i}=0\right\}_{i\in I_2}\\&\left\{xX^{C_i}\neq 0\right\}_{i\in I_3}\\&\left\{x'X^{D_i}\neq 0\right\}_{i\in I_4}\end{aligned}$$

has a solution in atomless BA iff all of the following conditions hold:

1. no A_i equals C_i,

2. no B_i equals D_i,

3. no X^{C_i}, X^{D_i} is zero,

4. $X^{A_i}=0$ whenever $A_i=B_j$,

In which case a solution is $x=\bigcup_j t_j\cup\bigcup_m X^{B_m}$ for any $0<t_i<X^{C_i}$.

PROOF. Necessity of 1,2,3,4 is immediate. For sufficiency we simply plug-in the solution:

$$xX^{A_i}=\bigcup_j t_jX^{A_i}\cup\bigcup_m X^{A_i}X^{B_m}=0$$

by 1,4.

$$\begin{aligned}x'X^{B_i}&=X^{B_i}\bigcap_j t_j'\bigcap_m X^{B_m\prime}\leq X^{B_i}X^{B_i\prime}=0\\xX^{C_i}&=\bigcup_j t_jX^{C_i}\cup\bigcup_m X^{B_m}X^{C_i}\geq t_i\neq 0\\x'X^{D_i}&=X^{D_i}\bigcap_j t_j'\bigcap_m X^{B_m\prime}=X^{D_i}\bigcap_j t_j'\neq 0\end{aligned}$$

where the second equality is by condition 2 and the third is because the complement of each t_j contains a nonzero part from each nonzero minterm. □

REMARK 4.1. In case there is no A_i, simply solve for x'.

REMARK 4.2. Note that this minterm normal form allows not only finding solutions but also an alternative method of quantifier elimination.

Remark 4.3. For a close-to-minimal solution (as strictly minimal usually doesn't exist), choose the smallest available t_i, and while transforming the system to minterm form, choose H_i s.t. $h_i(H_i)$ is the smallest.

Remark 4.4. theorem 3.1 and its proof may also be used, and is in fact very similar to the latter theorem. It also explicitly handles BFs rather SBFs.

4.2. In Order Normal Form

cf. section 2.3.3 for the setting discussed here.

Theorem 4.3. *In atomless BA, there exists x s.t.*

$$a \le x \le b$$

$$\{c_i \nleq x\}_{i \in I}$$

$$\{x \nleq d_j\}_{j \in J}$$

iff for all i, j:

$$c_i \nleq a \le b \nleq d_j$$

Proof. Exercise. □

Remark 4.5. Note that $c_i \nleq a \le b \nleq d_j$ reads $c_i \nleq a \wedge a \le b \wedge b \nleq d_j$. It does not mean, for example, that $a \nleq d_j$.

Remark 4.6. In this formulation we assume that the positive condition $a \le x \le b$ always appears, even if only $0 \le x \le 1$.

In light of the previous section, to find an explicit solution it is enough to find one for a system without a positive part. Sometimes a "nice" solution exists:

Lemma 4.1. *In any BA (not necessarily atomless), if the system*

$$\{c_i \nleq x\}_{i \in I}$$

$$\{x \nleq d_j\}_{j \in J}$$

where

$$\forall_{i \in I} \forall_{j \in J} . c_i d_j' = 0$$

has a solution, then if $I = \emptyset$ then $x = 1$ is a solution, and if $J = \emptyset$ then $x = 0$ is a solution, otherwise

$$x = \bigcup_{j \in J} d_j'$$

is a solution.

Proof. The case of empty I, J is trivial. For the general case, a solution exists iff $c_i \neq 0 \wedge d_j \neq 1$. Now simply

$$d_i' x = d_i' \bigcup_{j\in J} d_j' \geq d_i' \neq 0$$

$$c_i x' = c_i \bigcap_{j\in J} d_j = c_i \neq 0$$

since $c_i d_j' = 0$ is same as $c_i \leq d_j$. $\square$

In the previous lemma, x is an SBF in C, D. This is not always the case, for example in $c < x < d$, no solution can be written as an SBF in c, d. However it is somewhat easy to classify all cases in which x is indeed an SBF in C, D, and moreover, it is easy to see that it is always possible to write a solution as a BF (since that BF can simply equal a constant which is a solution). We start with the following lemma which in particular pins down the systems in which such an SBF exists:

Lemma 4.2. *In a system $\{c_i \nleq x\}_{i\in I} \wedge \{x \nleq d_j\}_{j\in J}$, a necessary and sufficient condition that $x = f(C, D)$, for some BF f, is a solution, is:*

(1) for all $i \in I$ exists $P_i \in \{0,1\}^{|I|}, Q_i \in \{0,1\}^{|J|}$ s.t. $p_i = 1$ and $f(P_i, Q_i) \neq 1$, and
(2) for all $j \in J$ exists $U_j \in \{0,1\}^{|I|}, V_j \in \{0,1\}^{|J|}$ s.t. $v_j = 0$ and $f(U_j, V_j) \neq 0$.

Proof. Write f in minterm normal form

$$x = f(C, D) = \sum_{A,B} f(A, B)\, C^A D^B$$

now trivially

$$x' c_i = \sum_{A,B} f'(A, B)\, c_i C^A D^B \neq 0 \to \exists AB.a_i = 1 \wedge f(A, B) \neq 1$$

$$x d_j' = \sum_{A,B} f(A, B)\, d_j' C^A D^B \to \exists AB.b_j = 0 \wedge f(A, B) \neq 0$$

$\square$

Definition 4.1. In a BA $\mathcal{B}$, a *splitter* is a partial function $S : \mathcal{B} \to \mathcal{B}$ s.t. $0 < S(x) < x$ for all x which is nonzero nonatom.

Clearly a splitter always exists in atomless BA. Henceforth we shall assume the existence of a splitter denoted by $\mathcal{S}$. We say that x has a *good splitter* if calculating $\mathcal{S}(x)$ does not make use of the atomless assumption, e.g. when x is explicitly written as $x = y \cup z$ with $yz \neq 0 \wedge y \neq z$. Otherwise we say that x has only a *bad splitter*.

The following lemma and corollary was obtained by [**pp**]:

LEMMA 4.3. *If the system* $\{c_i \nleq x\}_{i \in I} \wedge \{x \nleq d_j\}_{j \in J}$ *has a solution, and if* x *satisfies*

$$\forall AB.C^A D^B \neq 0 \rightarrow xC^A D^B \neq 0 \wedge x' C^A D^B \neq 0$$

alternatively

$$x = \bigcup_{A,B} \mathcal{S}\left(C^A D^B\right)$$

then x *is a solution.*

PROOF. Simply

$$x' c_i = x' \left(c_i \bigcup_{A,B} C^A_{-i} D^B \right) \geq x' c_i C^A_{-i} D^B \neq 0$$

$$x d'_i = x \left(d'_i \bigcup_{A,B} C^A D^B_{-i} \right) \geq x' d'_i C^A D^B_{-i} \neq 0$$

where C^A_{-i} refers to a minterm in all c's except c_i, and where the last inequalities follow from the fact that the system has a solution, so $c_i \neq 0$ therefore at least one minterm with c_i appearing positively is nonzero, and similarly for d_j. □

COROLLARY 4.1. *If the system* $\{c_i \nleq x\}_{i \in I} \wedge \{x \nleq d_j\}_{j \in J}$ *has a solution, and if* x *satisfies*

$$x = \bigcup_{A,B \in \mathcal{T}} \mathcal{S}\left(C^A D^B\right)$$

when $\mathcal{T}$ *is a set of pairs of bitstrings s.t. containing each* c_i *positively at least once and each* d_j *negatively at least once, and s.t.* $C^A D^B$ *is nonempty for all* $A, B \in \mathcal{T}$, *then* x *is a solution. Moreover, such an* x *always exists.*

PROOF. Fully along the lines of the previous proof. □

REMARK 4.7. Finding $\mathcal{T}$ can be done in quadratic time using a simple greedy algorithm: start with c_1 and conjunct it with c_2 and c'_2, in parallel, and similarly for d. Proceed with the nonempty branch and continue.

LEMMA 4.4. *In atomless BA, if the system*

$$a \leq x \leq b$$
$$\{c_i \nleq x\}_{i \in I}$$
$$\{x \nleq d_j\}_{j \in J}$$

has a solution, then a minimal solution exsits iff $\forall j.a \nleq d_j$, in which case $x = a$ is the minimal solution. Similarly a maximal solution exists iff $\forall i.c_i \nleq b$, in which case $x = b$ is the maximal solution.

PROOF. Exercise. □

REMARK 4.8. cf. remark 4.5.

4.3. The Multivariate Case

The system

$$f(X, Y) = 0$$
$$\{g_i(X, Y) \neq 0\}_{i \in I}$$

(where X, Y are tuples of variables) can be solved by eliminiating each single variable at a time. However we can use the LGRS in order to eliminate whole X at once. Let $Z(Y)$ be such that $f(Z(Y), Y) = 0$ (which can be found using theorem 4.1 wrt X and obtaining a solution that depends on Y). Then by LGRS we can write the system as

$$\bigcap_{A,B} f(A, B) = 0$$
$$\{g_i(Z(Y) f(X) + X f'(X), Y) \neq 0\}_{i \in I}$$

which can then be converted into

$$\bigcap_{A,B} f(A, B) = 0$$
$$\left\{\bigcup_{A} g_i(Z(Y) f(A) + A f'(A), Y) \neq 0\right\}_{i \in I}$$

where A run over all $0, 1$ vectors.

CHAPTER 5

Decidable Conservative Extensions

In this chapter we will show how to extend the theory of BA with additional constructs, and how to reduce those extensions back to the pure theory of BA.

5.1. Cardinality

In what follows $f(x) = ax + bx'$ is any Boolean function.

THEOREM 5.1. *Let $f(x)$ be a Boolean function. Then its range is the interval $[ab, a \cup b]$.*

PROOF. Exercise. $\square$

COROLLARY 5.1. *The equation $|f(x)| = n$ has a solution iff $|ab| \le n \le |a \cup b|$.*

The following theorem is a strong and useful generalization of Boole's consistency condition:

THEOREM 5.2. *Let $f(x)$ be a Boolean function. Then the minimum of $|f(x)|$ is attained precisely when*

$$a'b \le x \le a' \cup b$$

and the maximum precisely when

$$ab' \le x \le a \cup b'$$

PROOF. By theorem 1 the minimum of $|f(x)|$ is $|ab|$. The set of x's s.t. $f(x) = ab$ is given by solving

$$g(x) = ax + bx' + ab = 0$$

and the general solution is $g(0) \le x \le g'(1)$ namely $a'b \le x \le a' \cup b$ and the claim for the minimum is proved. the For maximum, similarly write

$$g(x) = ax + bx' + a \cup b = 0$$

so $b + a \vee b \le x \le a' + a \vee b$ equivalently $ab' \le x \le a \vee b'$. $\square$

5.2. Cartesian Product

Given an expression involving $\cup, \cap, '$, $\times$ and constants and variables, where $\times$ is interpreted over the sets underlying the BA elements (as guaranteed by Stone's representation theorem for BAs, alternatively over any BA interpreted over fixed sets), and whenever this expression typechecks so cartesian product of e.g. two elements cannot interact as-is with a cartesian product of e.g. three elements, we can use the well known identities

$$(ab) \times (cd) = (a \times c)\,(b \times d)$$

$$(a \times b)' = (a' \times b') \cup (a \times b') \cup (a' \times b)$$

(or similar identities widespread in literature) to push $\times$ to the innermost level in the expression. Then given a first order formula, we can make the BF appearing in each atomic formula take e.g. the form of disjunctions of cartesian products of minterms. We now convert the formula to minterm normal form (or a weaker form based on DNF of BFs). Now pulling out $\times$ over the conjunctions in each clause, we know that the product equals the empty set iff at least one multiplicand is empty, which would be a disjunction of formulas without $\times$.

Note that this allows cartesian product of elements from different BAs as in the many-sorted theory of BAs.

5.3. Higher-Order Boolean Functions

It is possible to quantify over BFs, SBFs, and certain CBFs (conditional BFs as below), and their higher order counterparts, and obtain an equivalent formula without quantification over functions, using the following method. Consider a formula involving existential (or universal, mutatis mutandis) quantification $\exists f$ over such functions. Each BF of n variables can be written as a Boolean expression involving 2^n constants (e.g. by using Boole's normal form or algebraic normal form or minterm normal form, per subexpression considering a single variable, or over the whole expression considering all variables), so quantification over BFs is converted into 2^n first order quantifiers. Similarly for SBFs we quantiy over constants and require them to be either 0 or 1. A CBF is a Boolean expression that involves the ceiling function defined by taking zero to zero and all other BA elements to one, or even more generally, a formula in the language of BA that is interpreted as the values 0,1 in the BA (which is the same as allowing quantifiers and equality/inequality under the ceiling function). In their full generality, CBFs may involve unboundedly many coefficients. Restricting them, e.g. by requiring that expressions under the ceiling function (or in formulas) must be SBFs, or requiring constants to be taken from some fixed finite set, allows a quantifier elimination into first order in the same fashion as above.

Higher order functions (BF, SBF, and restricted CBF) are seen as operating over the coefficients of their input (possibly higher order) functions and returning coefficients, and are therefore translated accordingly, so a higher order function that takes a BF of n variables and returns a BF of n variables, will be written as a function that takes 2^n BA elements and returns 2^n elements, with all necessary adjustment for all cases, mutatis mutandis, and similarly for a function that takes a function of functions, and so on.

For efficiency, there is no need to expand the formula exponentially (or a tower of exponentials) right at the beginning, but it can be done step-by-step with opportunities for simplifications and eliminations in each step, in the following fashion: a quantifier over a BF of n variables can be converted to a quantification over two BA elements and over two BFs over $n-1$ variables, simply by writing down the Boole's normal form (or any other form e.g. Reed-Muller) for the quantified function w.r.t. one (possibly cleverly chosen) variable.

5.3.1. Application to Second Order Finite Model Checking. TBD: obviously second order finite model checking can be rewritten as quantification over SBFs.

5.4. Homomorphisms and Hemimorphisms

In what follows we will deal with existential formulas of the form

$$\phi \equiv \exists x_1, \dots x_n . f(X) = 0 \wedge \bigwedge_j g_i(X) \neq 0 \wedge \bigwedge_i \psi_i$$

where each ψ_i is of the form $x = h_j(y)$ where x, y may be constants, or taken from $x_1, \dots x_n$. h_j here is either a BA homomorphism or a monoid homomorphism (as we shall describe shortly), and the rest of ϕ is the general form of a DNF clause in the language of BA. We will transform ϕ to a formula which does not contain h_j.

Remark 5.1. Here we support the many-sorted theory of BA, so it is interpreted in the product of multiple BAs, and the homomorphisms may be between different BAs. In particular we can support ultrafilters which are nothing but homomorphisms into the two-element BA.

A homomorphism here is simply a ring homomorphism. The term hemimorphism is used by Halmos and is defined by:

DEFINITION 5.1. A function $h : \mathcal{B}_1 \to \mathcal{B}_2$ between two BAs is a *hemimorphism* if $h(0) = 0$ and $h(x \cup y) = h(x) \cup h(y)$ for all x, y.

Any hemimorphism gives rise to a monoid homomorphism, where the monoid is the multipicative monoid in the BR. Put $g(x) = h'(x')$. Then $g(1) = 1$ and

$$g(xy) = h'(x' \cup y') = (h(x') \cup h(y'))' = h'(x')\, h'(y') = g(x)\, g(y)$$

This is the same as existential and universal quantifiers in description logic, where h is seen as a binary relation, and BA elements are seen as unary relations. We will emphasize on this connection later on.

We therefore assume that each h_j in the original formula is either a homomorphism or a hemimorphism, which includes the case of monoid homomorphism. Further, we can also cover isomorphisms, by requiring that a homomorphism has an empty kernel and that it sends 1 to 1, by a modification of the technique below.

First we convert $\bigwedge_i \psi_i$ into the form

$$\bigwedge_{(i,j,k)\in I} \left[\bigcup_{A\in\mathcal{A}_i} c_A X^A\right] = h_j\left(d_k X^{B_k}\right)$$

where c_A, d_k are constants. This translation is straight-forward by writing each element as a disjoint union of minterms, and relying on the fact that h_j distributes over unions.

We now got a finite partition of the BA where the disjoint parts are the minterms. We can walk over the graph defined by which minterm is sent to which. The only additional condition we have to add is

$$d_k X^{B_k} = 0 \to \bigcup_{A\in\mathcal{A}_i} c_A X^A = 0$$

with the initial condition dictated by $f(X)$ saying which minterms must be zero, and this readily comes down to a method to eliminate the hemimorphisms.

(TBD: fix till the end of the section) For homomorphisms we add the following condition: disjoint elements are sent to disjoint elements, namely $xy = 0 \to h(x)\, h(y) = 0$. This can again be checked by walking on the graph of which minterm is sent to which.

However in BAs that are not atomless, another cardinality condition has to be added. This and other results required for those algorithms are summarized in the following theorem:

THEOREM 5.3. *If $x_1, \ldots, x_n$ are nonzero and disjoint then there is a hemimorphism h s.t. $\forall i.y_i = h(x_i)$ for arbitrary $y_1, \ldots, y_n$. Under* *the same setting, and if the BA is complete or countable atomless, a homomorphism exists iff $y_i y_j = 0$ for all $i \neq j$, and $|x_i| \leq |y_i|$.*

Remark 5.2. $|x|$ refers to cardinality, and in pure BA terms, it is the supremum of how many disjoint sets x can be written as a union thereof.

Proof. Set h to send anything in $[\bigcup_i x_i]'$ to zero, and for hemimorphisms for all $0 < t_i \leq x_i$,set $h(t_i) = y_i$. The rest is immediate. For homomorphisms, if the BA is complete then this follows from theorem 5.13 in [**kop**]. we use Stone's duality in its topological setting, recalling that a homomorphism is the [set] inverse of continuous functions (in the Stone topology), and vice versa. We have to find a continuous function f s.t. $\forall i . y_i = f^{-1}(x_i)$. But this already says that certain clopen sets are sent to clopen sets, and disjoint sets are sent to disjoint sets, so as long as the preimage of each set is not smaller (in terms of cardinality) than the original set (and in atomless BA all clopen sets are infinite), there exists an continuous extension of this function over the whole space. In particular we can again set h to send anything in $[\bigcup_i x_i]'$ to zero. ☐

Remark 5.3. In the above algorithms, the cardinality constraint has to be clearly addressed, e.g. by not fixing the underlying BA and allowing it to be infinite (which will require a careful consideration of the constants), or by considering an atomless BA so the cardinality of each element is either zero or infinite.

5.5. Converse Algebras

Relation Algebras were extensively studied by Tarski. The intuition behind them is to study the BA $\mathcal{P}(X \times X)$ (or any subalgebra thereof) over some set X. It is a BA of binary relations extended with additional operators, in particular composition and converse. We will deal here with what we refer to as converse algebras (CA), so no composition is involved, and the converse of a binary relation R^- is defined by $\forall xy . Rxy \leftrightarrow R^- yx$. It is possible to give a more general and abstract definition of converse, and even abstract the underlying BA from $\mathcal{P}(X \times X)$ or its subalgebras, but we shall not deal with it here. We will just distinguish one case, which we shall refer to as diagonal-free converse algebras (DFCA). It means that we treat binary relations while ignoring their diagonal, so they never contain pairs of the form Rxx. For this we only need to treat negation: when we take the complement of a relation we make sure to remove the diagonal as well, so $R' \equiv (X \times X)_{-d} \backslash R$.

We use the following notation: R_d will denote the diagonal of R. R_{-d} will denote R without its diagonal, so $R_{-d} = R\,[R_d]'$. $R_s = RR^-$ denotes the symmetric part of R, while $R_a = RR^{-\prime}$ denotes its asymmetric part.

A DFCA is *complete* if every relation R has a maximal asymmetric part, so $\forall R \exists T. R \cup R^- = T \cup T^- \wedge T \subseteq R \wedge TT^- = 0$.

5.5.1. Zeros of Polynomials. A converse polynomial in R will be a BF in R_d, R, R^-, and over DFCA can be written as

$$f\left(R, R^-\right) = ARR^- + BRR^{-\prime} + CR'R^- + DR'R^{-\prime}$$

while in general CA we will use

$$f\left(R_d, R, R^-\right) = ARR^-R_d' + BRR^{-\prime} + CR'R^- + DR'R^{-\prime} + ER_d$$

THEOREM 5.4. *In a complete DFCA, a converse polynomial has a zero iff*

$$\left(A \cup A^-\right)\left(B \cup C^-\right)\left(B^- \cup C\right)\left(D \cup D^-\right) = 0$$

PROOF. Clearly $f\left(R, R^-\right) = 0$ iff $f\left(R, R^-\right) \cup f^-\left(R, R^-\right) = 0$, and

$$f\left(R, R^-\right) \cup f^-\left(R, R^-\right)$$

$$= \left(A \cup A^-\right) RR^- + \left(B \cup C^-\right) RR^{-\prime} + \left(C \cup B^-\right) R'R^- + \left(D \cup D^-\right) R'R^{-\prime}$$

so

$$\begin{aligned} RR^- &\leq A'A^{-\prime} \\ RR^{-\prime} &\leq B'C^{-\prime} \\ R'R^- &\leq C'B^{-\prime} \\ D \cup D^- &\leq R \cup R^- \end{aligned}$$

Observe that the second and third equations are the same by taking the converse on both sides. Noting that $R \cup R^- = RR^- \cup R'R^- \cup RR^{-\prime}$, so

$$D \cup D^- \leq R \cup R^- \leq A'A^{-\prime} \cup B'C^{-\prime} \cup C'B^{-\prime}$$

therefore necessary condition for the existence of solution is

$$D \cup D^- \leq A'A^{-\prime} \cup B'C^{-\prime} \cup C'B^{-\prime}$$

alternatively

$$\left(A \cup A^-\right)\left(B \cup C^-\right)\left(B^- \cup C\right)\left(D \cup D^-\right) = 0$$

To show that this is also sufficient, take any

$$\left(A'A^{-\prime}T_1^{-\prime} \cup T_1\right)\left(D \cup D^-\right) \leq R \leq A'A^{-\prime} \cup T_2$$

where T_1, T_2 are maximal asymmetric parts of $B'C^{-\prime}$. □

The more general case is treated similarly:

THEOREM 5.5. *In a complete CA, a converse polynomial has a zero iff*

$$\left(A\cup A^{-}\right)\left(B\cup C^{-}\right)\left(B^{-}\cup C\right)\left(D\cup D^{-}\right)=0$$
$$D_dE_d=0$$

PROOF. Write

$$0=f\left(R_d,R,R^{-}\right)\cup f^{-}\left(R_d,R,R^{-}\right)$$

$$=\left(A\cup A^{-}\right)RR^{-}R_d'+\left(B\cup C^{-}\right)RR^{-\prime}+\left(C\cup B^{-}\right)R'R^{-}+\left(D\cup D^{-}\right)R'R^{-\prime}+E_dR_d$$

so

$$\begin{gathered}RR^{-}R_d'\leq A'A^{-\prime}\\ RR^{-\prime}\leq B'C^{-\prime}\\ R'R^{-}\leq C'B^{-\prime}\\ D\cup D^{-}\leq R\cup R^{-}\\ R_d\leq E_d'\end{gathered}$$

implying $\left(D\cup D^{-}\right)_d\leq E_d'$ which is same as $D_d\leq E_d'$. The rest of the conditions are same as in the DFCA case. A solution would then be $R=A'A^{-\prime}\cup D_d\cup T$ where T is again a maximal asymmetric part of $B'C^{-\prime}$. □

5.5.2. Query Answering. Here we shall define a somehow non-standard notion of query answering. In the field of Knowledge Representation (KR) a query would be an open formula, and the answer would be all substitutions that are *entailed* from the KB. Another way to say it, is that it refers to the part that is common to all models. So if the query is merely an atom of the form Rxy, then the answer resembles $\bigcap_{M\models KB}M$. There are some caveats here but the main takeaway is concerning the part that is common to all models indeed. Note that this need not be a model: consider the formula

$$C\left(a\right)\wedge\left(C\left(b\right)\vee C\left(c\right)\right)$$

where C is a unary relation and a,b,c are constants. Then the part common to all models is only $C\left(a\right)$, however it is not a model, since every model will have to include either $C\left(b\right)$ or $C\left(c\right)$.

Our modified notion of query answering is as follows. Initially, the query is a single atom of the form Rxy. The answer is going to be a formula with two free variables, in which R does not appear, and for each substitution of the variables, the formula is a tautology iff that substitution holds in all models of R. For example, the answer to the query Rxy over each of the two formulas

$$\forall xy.Sxy\rightarrow Rxy$$

$$\forall xy.Sxy\leftrightarrow Rxy$$

is going to be Sxy. An intuitive way to look at it is that the answer gives an "explanation" that "explains" R without referring to R.

We model the KB as a statement of the form $f(R, R^-) = 0$. The reason for that will be clear in later chapters. The coefficients $A, B, C, D\,[, E]$ may depend on other variables and constants. We would like to express the query answer α being an R-free expression satisfying

$$\alpha = \bigcap_{R|f(R,R^-)=0} R$$

The following theorem was obtained with help from [**pp**]:

THEOREM 5.6. *In a DFCA, if* $\exists R.f(R, R^-) = 0$*, then*

$$\alpha = \left(D \cup D^-\right)\left(B^- \cup C\right)$$

PROOF. Assume $(x, y) \in (D \cup D^-) \backslash C'B^{-\prime}$. So $(y, x) \notin B'C^{-\prime}$ therefore $(y, x) \notin R_a; (x, y) \notin R_a^-$. Since $(x, y) \in D \cup D^-$, then $(x, y) \in R_s \cup R_a$ which reads $(x, y) \in R$.

For the other direction, if $(x, y) \notin (D \cup D^-) \backslash C'B^{-\prime}$ then either

1. $(x, y) \notin D \cup D^-$ therefore $(y, x) \notin D \cup D^-$ so there exists a solution satisfying $(x, y) \notin R$.

2. $(x, y) \in C'B^{-\prime} \cap (D \cup D^-)$, if there is a model with $(x, y) \in R$, then take a model with $(x, y) \notin R$ but $(y, x) \in R$. $\square$

The CA case is completely analogous and gives the answer

$$\alpha = D_d \cup \left(D \cup D^-\right)_{-d}\left(B^- \cup C\right)$$

5.6. Monadic Algebras

Monadic BAs are BAs with additional operator $\exists$ that satisfies the axioms:

- $\exists 0 = 0$
- $x \leq \exists x$
- $\exists (x \cup y) = \exists x \cup \exists y$
- $\exists (x \exists y) = \exists x \exists y$

Typically a BA may be equipped with many different such operators. We also denote $\forall = \neg\exists\neg$. For a good treatment of this subject cf. [**hal**]. In particular he shows that

$$x \leq \exists y \to \exists x \leq \exists y$$

and

$$x \leq y \to \exists x \leq \exists y$$

We shall use those facts. We are interested in solving equations involving $\exists$. One reason it is interesting is because LTAs are a main example of BAs but quantification is not a Boolean operations. We'd like to model and solve problems that involve quantification as well.

The following theorem was obtained with help from [**pp**]:

THEOREM 5.7. *For bivariate Boolean f, the equation*

$$f(x, \exists x) = 0$$

explicitly

$$ax\exists x + bx\neg\exists x + c(\exists x)\neg x + d(\neg x)(\neg\exists x) = 0$$

has a solution iff

$$(\exists d)(ac \cup \forall a) = 0$$

in which case, $x = a'\exists d$ is a solution.

PROOF. Since $x \leq \exists x$ we can write

$$ax + c(\exists x)\neg x + d\neg\exists x = 0$$

which reads

$$x \leq a'$$
$$c(\exists x) \leq x$$
$$d \leq \exists x$$

Now $d \leq \exists x$ implies $\exists d \leq \exists x$. $\exists d \leq \exists x$ implies $c\exists d \leq c\exists x$ therefore using the first and second equation, $c\exists d \leq x \leq a'$ so one necessary condition $ac\exists d = 0$ is established. The second necessary condition is $\exists d \leq \exists(a')$ since

$$(x \leq a' \wedge \exists d \leq \exists x) \rightarrow \exists d \leq \exists x \leq \exists a'$$

For sufficiency, set $x = a'\exists d$, so

$$a'\exists d \leq a'$$
$$c(\exists a'\exists d) \leq a'\exists d$$
$$d \leq \exists a'\exists d = \exists d$$

The first equation is trivial, the second follows from the first necessary condition, and the last follows from the second necessary condition. □

REMARK 5.4. A quantifier s.t. $\exists x = 0$ if $x = 0$ and otherwise $\exists x = 1$, is called a *simple quantifier* and is treated in [**hal**]. Enhancing the language of BA with such operator is trivial: convert any atomic formula of the form $f(x, \exists x) = 0$ into

$$[x = 0 \rightarrow f(0, 0) = 0] \wedge [x \neq 0 \rightarrow f(x, 1) = 0]$$

$$c_i(\exists_1 x, \exists_2 x) \nleq x$$
$$x \nleq d_j(\exists_1 x, \exists_2 x)$$

5.7. Infinitary Operations

The goal of this section is to present a method to explicitly evaluate the expressions

$$\bigcup_{X|\phi(X)} f(X)$$

and

$$\bigcap_{X|\phi(X)} f(X)$$

where X is a tuple of variables, f is a BF, and ϕ is a GSBE with X as its unknowns. We focus on atomless BAs, while the treatment for general BAs is analogous yet more complex, as will be seen from some of the general following lemmas.

The method presented here is in particular useful for the first order theory of BA (possibly interpreted in a specific BA with a dedicated constant symbol to each element) enhanced with the above operations, while maintaining decidability, by reducing it to the standard BA theory.

It is already surprising that BAs, in particular atomless BAs, are even closed under the above [possibly] infinitary operations. As we shall see, the results take an even more surprisingly simple form.

Clearly, it is enough to compute $\bigcup_{X|\phi(X)} f(X)$ since

$$\bigcap_{X|\phi(X)} f(X) = \left[\bigcup_{X|\phi(X)} f'(X)\right]'$$

and vice versa. Further, note that

$$\begin{aligned}
&\bigcup_{x_1,\ldots,x_n|\phi(x_1,\ldots,x_n)} f(x_1,\ldots,x_n) \\
&= \bigcup_{x_1|\phi(x_1,\ldots,x_n)} \bigcup_{x_2,\ldots,x_n|\phi(x_1,\ldots,x_n)} f(x_1,\ldots,x_n) \\
&= \bigcup_{x_2,\ldots,x_n|\phi(x_1,\ldots,x_n)} \bigcup_{x_1|\phi(x_1,\ldots,x_n)} f(x_1,\ldots,x_n)
\end{aligned}$$

where in the last two equations x_1 cleary depends on $x_2,\ldots,x_n$. This shows that treating only the univariate case $\bigcup_{x|\phi(x)} f(x)$ is sufficient.

For simplicity, all BAs in this section are assumed to be infinite. The finite case treatment can be done along the same lines. However at one point we'll strongly use the atomless assumption, in which case our main and final result, under a mild assumption on f (otherwise the answer is trivial), is

$$\bigcup_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} h(x) = h(1)\,f'(1)\cup h(0)\,f'(0)$$

which, most remarkably, is not only a simple closed-form, but also does not depend on g_i. The intuition behind the latter point will be clear later on.

Lemma 5.1. *Let $a\in\mathcal{B}$, then*

$$\bigcup_{x\nleq a} x = \begin{cases} 0 & a=1 \\ 1 & a\neq 1 \end{cases}$$

$$\bigcap_{x\nleq a} x = \begin{cases} 1 & a=1 \\ a' & a' \text{ is an atom} \\ 0 & \text{otherwise} \end{cases}$$

Proof. We assume that empty disjunction are 0 and empty conjunctions are 1. For the disjunction claim, the case $a=1$ is trivial. The case of $a\neq 1$ is also trivial since then $1\nleq a$. For the conjunction claim, $a=1$ is again trivial. Suppose a' is a nonatom. Write $a'=b\vee c$ where b,c are nonzero and $bc=0$. So $b\nleq a$ and $c\nleq a$. Therefore $\bigcap_{x\nleq a} x\leq bc=0$. Now suppose a' is an atom. Then $x\nleq a$ iff $a'\nleq x'$ iff $a'\leq x$ by proposition 2.4. So $a'\leq\bigcap_{x\nleq a} x$. But $a'\nleq a$ so $\bigcap_{x\nleq a} x\leq a'$ therefore $\bigcap_{x\nleq a} x=a'$. □

Lemma 5.2. *Let $a\in\mathcal{B}$, then*

$$\bigcup_{x\ngeq a} x = \begin{cases} 0 & a=0 \\ a' & a \text{ is an atom} \\ 1 & \text{otherwise} \end{cases}$$

$$\bigcap_{x\ngeq a} x = \begin{cases} 1 & a=0 \\ 0 & \text{otherwise} \end{cases}$$

Proof. If $a=0$ then the disjunction and the conjunction are empty. If $a=1$ then all $x\neq 1$ satisfy $x\ngeq a$. Suppose a,a' are both non-atoms. Then exist c,d s.t. $0<c<a\wedge 0<d<a'$. Set $x=c\vee d$. Then $x\ngeq a$ since and $x'\ngeq a$, since $(c\vee d)\,a=c\neq 0$ and

$$ac'd'=ac'd+ac'=acd+ad+ac+a=a+c\neq 0$$

therefore in this case, the big union is one and the big intersection is zero. If a is an atom, we can write the two equations as

$$\bigcup_{x\in\mathcal{B}} a'x$$

$$\bigcap_{x\in\mathcal{B}} a'x$$

which, by theorem 2.3, equal a' and 0, respectively. If a' is an atom we note that

$$\bigcup_{x\ngeq a} x = \bigcup_{x'\nleq a'} x = \bigcup_{x\nleq a'} x' = \left[\bigcap_{x\nleq a'} x\right]' = 1$$

where the last equality is by 5.1, similarly

$$\bigcap_{x\ngeq a} x = \left[\bigcup_{x\nleq a'} x\right]' = 0$$

□

LEMMA 5.3. *Let f be a BF and $a \in \mathcal{B}$. Then*

$$\bigcup_{x\nleq a} f(x) = \begin{cases} 0 & a = 1 \\ af(0) \cup f(1) & a' \textit{ is an atom} \\ f(0) \cup f(1) & \textit{otherwise} \end{cases}$$

$$\bigcap_{x\nleq a} f(x) = \begin{cases} 1 & a = 1 \\ f(1)(f(0) \cup a') & a' \textit{ is an atom} \\ f(0) f(1) & \textit{otherwise} \end{cases}$$

$$\bigcup_{x\ngeq a} f(x) = \begin{cases} 0 & a = 0 \\ f(0) \cup a'f(1) & a \textit{ is an atom} \\ f(0) \cup f(1) & \textit{otherwise} \end{cases}$$

$$\bigcap_{x\ngeq a} f(x) = \begin{cases} 1 & a = 0 \\ f(0)(f(1) \cup a) & a \textit{ is an atom} \\ 0 & \textit{otherwise} \end{cases}$$

PROOF. Write $f(x) = xf(1) \cup x'f(0) = (f(1) \cup x')(f(0) \cup x)$. Then, using 5.2 and 5.1:

$$\bigcup_{x\nleq a} f(x) = \bigcup_{x\nleq a} xf(1) \cup \bigcup_{x\nleq a} x'f(0) = f(1) \bigcup_{x\nleq a} x \cup f(0) \bigcup_{x\nleq a} x'$$

$$= f(1)\bigcup_{x\nleq a} x \cup f(0)\left[\bigcap_{x\nleq a} x\right]' = f(1)\begin{cases}0 & a=1\\ 1 & a\neq 1\end{cases}\cup f(0)\cdot\begin{cases}0 & a=1\\ a & a' \text{ is an atom}\\ 1 & \text{otherwise}\end{cases}$$

similarly

$$\bigcup_{x\ngeq a} f(x) = f(1)\bigcup_{x\ngeq a} x \cup f(0)\left[\bigcap_{x\ngeq a} x\right]'$$

$$= \left[f(1)\cap\begin{cases}0 & a=0\\ a' & a \text{ is an atom}\\ 1 & \text{otherwise}\end{cases}\right]\cup\left[f(0)\cap\begin{cases}0 & a=0\\ 1 & \text{otherwise}\end{cases}\right]$$

and the intersections are dual, e.g. $\bigcap_{x\ngeq a} f(x) = \left[\bigcup_{x\ngeq a} f'(x)\right]'$. □

Lemma 5.4. *Let $a_1,\dots,a_n$ be elements of $\mathcal{B}$, none of which is 1, and where $\mathcal{B}$ is atomless, and $n>1$. Put $X=\{x|\,(x\nleq a_1)\wedge\cdots\wedge(x\nleq a_n)\}$. Then*

$$\bigcup_{x\in X} x = 1$$

and

$$\bigcap_{x\in X} x = 0$$

Proof. Write X as $X=\{x|\,(a_1'x\neq 0)\wedge\cdots\wedge(a_n'x\neq 0)\}$. Let

$$Y=\{x|\,(a_1'x\neq 0)\wedge\cdots\wedge(a_n'x\neq 0)\wedge(a_1'x'\neq 0)\wedge\cdots\wedge(a_n'x'\neq 0)\}$$

If we show that Y is nonempty then the lemma is proved because then X contains an element together with its complement, but the nonemptyness of Y follows directly from corollary 3.2. □

Remark 5.5. An interesting property of atomless BA arises from the proof. In any BA, if $f(x)=0$ then $f(x')\neq 0$ unless $f\equiv 0$. This is because $f(x)\cup f(x')=f(0)\cup f(1)$ for all f,x. However in case of $g_1(x)\neq 0,\dots g_n(x)\neq 0$,in atomless BA, there's always a satisfying x s.t. x' also satisfies the inequations.

Corollary 5.2. *For BFs f,g s.t. $f(0)f(1)=0$ (trivial otherwise) we have*

$$\bigcup_{x|f(x)=0} g(x) = g(f(0))\cup g(f'(1))$$

$$\bigcap_{x|f(x)=0} g(x) = g(f(0))\cap g(f'(1))$$

Proof. Direct application of theorem 2.9 and theorem 2.3. □

REMARK 5.6. Now $\bigcup_{x|f(x)\neq 0} g(x)$ can easily be evaluated by noting that

$$\bigcup_{x|f(x)\neq 0} g(x) = \bigcup_{x|x\ngeq f(0)} g(x) \cup \bigcup_{x|x\nleq f'(1)} g(x)$$

and using 5.3.

COROLLARY 5.3. *In atomless BA and for BFs f, g s.t. $f(0)f(1) = 0$ (otherwise use 5.3) we have*

$$\bigcap_{x|f(x)\neq 0} g(x) = g(0)\, g(1)$$

PROOF.

$$\bigcap_{x|f(x)\neq 0} g(x) = \left(g(1) \cup \left[\bigcup_{x|f(x)\neq 0} x \right]' \right) \left(g(0) \cup \bigcap_{x|f(x)\neq 0} x \right)$$

and use lemma 5.4. □

DEFINITION 5.2. A univariate BF f is called *wide* if $f(0)\, f(1) = 0$ and $\frac{\partial f}{\partial x} \neq 0 \bigwedge \frac{\partial f}{\partial x} \neq 1$.

Note that this means that f has more than one zero and is not identically zero. Clearly we can define "wide wrt x" in case f depends on several variables. Recall that the Boolean derivative is $\frac{\partial f}{\partial x} = f(0) + f(1)$.

DEFINITION 5.3. Let $\mathcal{B}$ be a BA and f a wide BF over it. Then $\mathcal{B}/f$ is the BA whose elements lie in the interval $[f(0), f'(1)]$. All BA operations are relative to this interval, so x' is $x' f'(1)$ and xy is $xy \cup f(0)$. We also define the epimorphism $h_f : \mathcal{B} \to \mathcal{B}/f$ by $h_f(x) = a \vee bx$.

COROLLARY 5.4. *Let $\mathcal{B}$ be countabe atomless and f a wide BF over it. Then $\mathcal{B}$ is isomorphic to $\mathcal{B}/f$.*

PROOF. It is easy to see that $\mathcal{B}/f$ is also countable and atomless. But all countable atomless BAs are isomorphic. □

The following is trivial:

COROLLARY 5.5. *Let $\mathcal{B}$ be countabe atomless and f a wide BF over it. The univariate elementary GSBE $f(x) = 0 \wedge \bigwedge_i g_i(x) \neq 0$ has a solution iff $\bigwedge_i g_i(x) \neq 0$ has a solution in $\mathcal{B}/f$.*

Corollary 5.6. *Let $\mathcal{B}$ be countabe atomless and f a wide BF over it. Let $g_1, \ldots, g_k$ be univariate BFs, none of which is identically zero. Then*

$$\bigcup_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} x = f'(1)$$

$$\bigcap_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} x = f(0)$$

Proof. Consider the expressions

$$\bigcup_{x|\bigwedge_i g_i\left(h_f(x)\right)\neq 0} h_f(x)$$

$$\bigcap_{x|\bigwedge_i g_i\left(h_f(x)\right)\neq 0} h_f(x)$$

which is the same as considering

$$\bigcup_{x\in{}^{\mathcal{B}}\!/_{f}|\bigwedge_i g_i(x)\neq 0} x$$

$$\bigcap_{x\in{}^{\mathcal{B}}\!/_{f}|\bigwedge_i g_i(x)\neq 0} x$$

while the firt readily equals 1 by lemma 5.4 and the second equals 0. However the preimage of 1 in ${}^{\mathcal{B}}\!/_{f}$ anything greater or equal to $f'(1)$. But obviously

$$\bigcup_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} x \leq f'(1)$$

therefore equality follows. Similarly the preimage of 0 is anything less than $f(0)$, but all x's satisfying $f(x) = 0$ are at least $f(0)$, so the second equality follows. □

Remark 5.7. In case that f is not wide, the result can immediately be calculated as the union is either empty or contains one element.

Corollary 5.7. *Let $\mathcal{B}$ be countabe atomless and f a wide BF over it. Let $g_1, \ldots, g_k$ be univariate BFs, none of which is identically zero, and h some arbitrary BF. Then*

$$\bigcup_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} h(x) = h(1)\, f'(1) \cup h(0)\, f'(0)$$

Proof. Simply

$$\bigcup_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} h(x)$$

$$= h(1) \bigcup_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} x \cup h(0)\left[\bigcap_{x|f(x)=0\wedge\bigwedge_i g_i(x)\neq 0} x\right]'$$

and using the previous corollary. □

5.8. Recurrence Relations

We propose the notion of weakly ω-categorical theory. Recall that ω-categorical theory is a first order theory in which all of its countable models are isomorphic. The Ryll-Nardzewski theorem says that this definition is equivalent to another definition: that up to logical equivalence, there are only finitely many formulas with a fixed number of free variables. This gives rise to defining weakly ω-categorical theories: those are theories for which the number of formulas with fixed number of variables and where the constants appearing in them are taken from a fixed finite subset of all constants in the language, up to logical equivalence, is finite. For the sake of this chapter, it does not matter whether or not the theory is interpreted in a fixed structure.

It is easy to see that the theory of atomless BA and of fixed finite BA, are both weakly ω-categorical (cf. remark 2.3). In what follows we shall deal only with those BA theories. However the construction in this section can be carried out into any weakly ω-categorical theory. Yet in BA we have an additional aspect not covered by this notion: the above principle holds not only for formulas but also for terms. Specifically, there are only finitely many BFs with prescribed finite set of constants and variables. One special case is exercise 11.1.

Definition 5.4. The *ceiling* operator is a function defined by

$$\lceil x\rceil = \begin{cases} 0 & x = 0 \\ 1 & x \neq 0 \end{cases}$$

Definition 5.5. A *Conditional Boolean Function (, CBF)* is a finite combination of constants and variables by means of Boolean operations and the ceiling operator.

Note that under the ceiling operator we can have a whole expression, namely a whole CBF. So for example

$$\lceil x + y\rceil xy$$

is a CBF which is equivalent to

$$\begin{cases} 0 & x=y \\ xy & x\neq y \end{cases}$$

The following should be obvious:

PROPOSITION 5.1. *An equivalent definition of CBF is a function of the form*

$$CBF := BF|\textit{if } \phi \textit{ then } CBF \textit{ else } CBF$$

where ϕ is any formula in the language of BA. In case of atomless or finite BAs, yet another equivalent definition is obtained by allowing the ceiling operator to accept a formula ϕ, returning 0 or 1 as whether the formula is false or true.

We are now ready to define a formula in the language of BA enhanced with recurrence relations. It is a list of the form

$$f_0^1\left(X\right)=F_0^1\left(X\right)$$
$$f_n^1\left(X\right)=F^1\left(f_{n-1}^{k_1^1}\left(X\right),f_{n-2}^{k_2^1}\left(X\right),\dots,\phi_n^{m_1^1}\left(X\right),\phi_{n-1}^{m_2^1}\left(X\right),\dots\right)$$
$$f_n^2\left(X\right)=F^2\left(f_{n-1}^{k_1^2}\left(X\right),f_{n-2}^{k_2^2}\left(X\right),\dots,\phi_n^{m_1^2}\left(X\right),\phi_{n-1}^{m_2^2}\left(X\right),\dots\right)$$
$$\dots$$
$$\phi_n^1\left(X\right)=\Phi^1\left(f_{n-1}^{p_1^1}\left(X\right),f_{n-2}^{p_2^1}\left(X\right),\dots,\phi_n^{q_1^1}\left(X\right),\phi_{n-1}^{q_2^1}\left(X\right),\dots\right)$$
$$\dots$$
$$\psi\left(f^1,f^2,\dots,\phi^1,\phi^2,\dots\right)$$

This messy definition is actually very simple. We simply define formulas (the ϕ's) and CBFs (the f's) by means of recurrence relations, which may mutually depend on each other. ψ is the "main" formula. For example:

$$f_0\left(x,y\right)=xy'$$
$$f_n\left(x,y\right)=\lceil x=y\rceil x+yf_{n-1}\left(y,x\right)$$
$$\phi_1\left(x,y\right)=x$$
$$\phi_2\left(x,y\right)=y$$
$$\phi_n\left(x,y\right)=\exists z.\psi_{n-1}\left(x,z\right)\wedge\phi_{n-1}\left(z,y\right)\wedge\phi_{n-2}\left(z,x\right)$$
$$\forall x\exists y\exists z.f\left(x,y\right)=0\wedge\phi\left(y,z\right)$$

Here $f\left(x,y\right)$ is understood naturally as expected: it is the limit that f_n converges into. Similarly for ϕ. Clearly it does not always converge, but it is easy to pin down all cases, as follows:

(1) The dependency of functions, formulas, and their initial conditions, has to be well founded. So f_n can't depend on g_n (or ϕ_n) in case that g_n (or ϕ_n) depend on f_n. However f_n can depend on g_{n-1} and so on. This comes down to verifying that a certain directed graph is acyclic.
(2) The initial conditions should also be sufficient to allow calculating f_n, ϕ_n for any given n.
(3) And most importantly: while calculating $f_1, f_2, f_3, \dots$ and $\phi_1, \phi_2, \phi_3, \dots$ we are guaranteed to find a loop, namely for some $n \neq k$, $\phi_n \equiv \phi_k$ (logical equivalence) and similarly for f. if $n = k-1$ then it is a fixed point and the result is well-defined. Otherwise we can proceed in virtually any fashion: either we return 0 or $\perp$, or the first recurring expression, or we enhance the language to incoroporate "fallbacks" that return a default answer in case of no fixed point.

This, together with the properties of the language, should be sufficient to show that BA with recurrence relations can be written in an equivalent form in pure BA without recurrence relations.

REMARK 5.8. Multi-indices recurrence relations are supported in the same manner, namely recurrence relations of the form

$$f_{n,k} = F\left(f_{n,k-1}, f_{n-1,k}, \dots\right)$$

REMARK 5.9. Clearly the same construction can be carried out while involving higher-order BFs.

5.9. Strong Normalization

Although not an extension, we present an algorithm for "fully" simplifying a quantifier-free formula (or a formula with quantifiers in case we can easily eliminate them e.g. in atomless BA), s.t. two formulas are logically equivalent iff they're syntactically the same. It is also very useful for manual inspection of formulas and drawing conclusions from them.

Given a formula ϕ, convert ϕ to minterm normal form so each atomic formula is of the form $c_i X^{A_j} = 0$. We shall make use of C-style if-then-else operator, so $\phi ? \psi : \chi$ reads $(\phi \to \psi) \wedge (\neg\phi \to \chi)$. We construct the normalized formula recursively as follows, while maintaining two lists of assumptions P, N where P contains atomic formulas (positive assumptions) and N contains negated atomic formulas (negative assumptions). The input to the algorithm is a formula ϕ and assumptions P, N which is initially empty. The algorithm is denoted by $\alpha\left(\phi, P, N\right)$

(1) Pick an atomic formula $c_iX^{A_j} = 0$. We now return a formula of the form $c_iX^{A_j} = 0?\psi : \chi$
(2) For each assumption in N with exponent A_j, if the coefficient of that assumption is $\leq c_i$, set $\phi := \bot$.
   (a) Otherwise compute the disjunction of all coefficients of all assumptions in P with exponent A_j (maintain this disjunction along the path in the algorithm's execution tree so there is no need to recompute everything each time). If this disjunction is $\geq c_i$, set $\phi := \top$.
   (b) Otherwise, in ϕ, replace all occurences of $c_iX^{A_j} = 0$ with $\top$ (which also means replacing $c_iX^{A_j} \neq 0$ with $\bot$) and obtain a formula γ. Set $\psi := \alpha\left(\gamma, P \cup \left\{c_iX^{A_j} = 0\right\}, N\right)$.
(3) For each assumption in P with exponent A_j, if the coefficient of that assumption is $\geq c_i$, set $\chi := \bot$.
   (a) Otherwise compute the disjunction of all coefficients of all assumptions in N with exponent A_j (maintain this disjunction along the path in the algorithm's execution tree so there is no need to recompute everything each time). If this disjunction is $\leq c_i$, set $\chi := \top$.
   (b) Otherwise, in ϕ, replace all occurences of $c_iX^{A_j} = 0$ with $\bot$ (which also means replacing $c_iX^{A_j} \neq 0$ with $\top$) and obtain a formula δ. Set $\chi := \alpha\left(\delta, P, N \cup \left\{c_iX^{A_j} \neq 0\right\}\right)$.

This algorithm is quite a strong normalizer but it still doesn't satisfy the requirement that two formulas are logically equivalent iff they're syntactically the same. To assure that, we need some canonical order over the minterms and constants across all formulas. But we also need a deeper modification. We need to replace the constants with minterms of those constants. So for example if there are two constants c, d, we replace each occurence of c with $cd \cup cd'$ and each occurence of d with $cd \cup c'd$. Then each atomic formula of the form $cX^A = 0$ becomes a conjunction of two atomic formulas $cdX^A = 0 \wedge cd'X^A = 0$. Then run the algorithm α. The following preparation is useful to *sometimes* avoid exponential overhead.

We construct the following *constants tree* T. It represents all minterms in all constants appearing in the formula. Suppose the constants (which are not $0, 1$) appearing in the formula are $c_1, \ldots, c_n$. A prior optional step is to construct a list telling which constant (or its complement) is a subset of which constant (or its complement), so the list have elements of e.g. the form $c_i \leq c_j'$. This might speed up the construction of the constants tree.

The constant tree has a redundant root labelled with the BA element 1. Its depth is 0. Each node of depth $n > 0$ has two outgoing edges labelled c_n, c'_n, and two successor nodes (respectively) labelled with the conjunction of the label of that node, with c_n, c'_n (respectively). So for two constants the tree looks like $1\left(c_1\left(c_1c_2, c_1c'_2\right), c'_1\left(c'_1c_2, c'_1c'_2\right)\right)$. We can easily see how this tree represents all minterms. Clearly, once a node's label is zero, there is no need to continue that tree.

This constants tree is useful not only for minterm normal form over constants, but also to optimize the execution of α.

It is also possible to tweak the algorithm s.t. no full conversion to minterm normal form is required. We modify step 1 as follows:

(1) Pick an atomic formula $f\left(X\right) = 0$. Find a $0, 1$ assignment A s.t. $f\left(A\right) \neq 0$ (most easy when f is in BDD form). Split the atomic formula into the conjunction $f\left(A\right) X^A = 0 \wedge f\left(X\right) + f\left(A\right) X^A = 0$. Now run the rest of algorithm α wrt the atomic formula $f\left(A\right) X^A \neq 0$, and with the modifications below.

In step 2.(b) instead of replacing $c_iX^{A_j} = 0$ with $\top$ in ϕ, we take any atomic formula of the form $g\left(X\right) = 0$ in ϕ and test whether $g\left(A\right) = f\left(A\right)$ (the same f, A from modified step 1). If yes, we replace $g\left(X\right) = 0$ with $g\left(X\right) + f\left(A\right) X^A = 0$. If $g\left(A\right) \neq f\left(A\right)$, we do not replace anything. Similarly in step 3.(b), if $g\left(A\right) = f\left(A\right)$ then replace $g\left(X\right) = 0$ with $\bot$. If $g\left(A\right) \neq f\left(A\right)$ we do nothing.

In case f, g do not take the exact same set of variables, we perform those steps for each $0, 1$ assignment to the non-mutual variables. We can do that by finding all solutions of $f\left(A\right) + g\left(A\right) = 0$. Again it is most easy in BDD form.

CHAPTER 6

The Coutable Atomless Boolean Algebra

6.1. Completion

Show that the completion of SBF, is the powerset of all bitstrings, quotiened by taking two sets to be equivalent iff they differ only by finitely many bitstrings. Show directly that this is an atomless BA.

6.2. Homomorphisms and Ultrafilters

Getting hold on all endomorphisms of a BA is basically impossible. Even if we consider only ultrafilters (which are endomorphisms with range being only $0, 1$), it is well known that there exists so-called non-principal ultrafilters in the BA $\mathcal{P}(\mathbb{N})$, but proving their existence (or even the existence of only one of them) requires a weak form of the axiom of choice, so they're inherently non-constructive. The situation in free BAs in much simpler. In this subsection we show how to easily pin down all endomorphisms and endo-hemimorphisms over the BA of all SBFs, which is free, and a countable atomless BA.

Here, SBFs will be a finite Boolean combination of the variables $x_1, x_2, \ldots$.

THEOREM 6.1. *Let $h : SBF \to SBF$ be a homomorphism. Then there exists a sequence of SBFs $f_1, f_2, \ldots$ s.t. $h(f)$ returns the simultanous substitution of all $x_1, x_2, \ldots$ in f with $f_1, f_2, \ldots$. And vice versa: any such substitution is a homomorphism.*

PROOF. Since h is a homomorphism, and f is a Boolean combination of variables, h distributes over the Boolean operations and acts directly on the variables. Therefore it is defined solely on how it acts on $x_1, x_2, \ldots$. If it replaces them with other SBFs, it is easy to see that it is a homomorphism indeed. □

THEOREM 6.2. *Any ultrafilter $h : SBF \to SBF$ is a homomorphism s.t. in the setting of the previous theorem, each $f_1, f_2, \ldots$ is either identically 0 or identically 1.*

PROOF. One direction is immediate: if the substitutions are only $0, 1$ then the range of h is $0, 1$. For the other direction, suppose x_i

is replaced with f_i which is not identically $0, 1$. Consider $h(f)$ where $f \equiv x_i$. Then $h(f)$ is not 0 nor 1. $\square$

COROLLARY 6.1. *The set of all ultrafilters over the BA of SBFs can be constructively identified with the set of infinite bitstrings.*

REMARK 6.1. Stone duality tells us that each homomorphism is the inverse of a continuous function (in the Stone topology). The inverse of an endomorphism in the countable atomless BA is way deeper than merely endomorphism. To see this, recall that the p-adic topology is the Stone space of a countable atomless BA. Now recall Mahler's theorem.

TODO: monomorphisms, epimorphisms, hemimorphisms (minterms)

CHAPTER 7

NSO: Nullary Second Order Logic

7.1. Overview

Consider the LTA of sentences in some logical language. This LTA is a BA that comes with the theory of BA. The sentence $\forall x \exists y. xy' = 0$ in the language of BA interpreted in some LTA, would mean "forall sentence x exists a sentence y s.t. x entails y". So we can immediately see how the BA theory of an LTA is a theory that speaks about sentences of some language, where those sentences are not accessible syntactically, but abstracted to merely BA elements.

Countable atomless (CA) BAs arise naturally in logical languages. cf. remark 2.4 and recall that almost all languages of interest are countable. All CA BAs are therefore isomorphic, and moreover, all atomless BAs are elementarily equivalent.

So, if we manage to take a language that makes a CA BA (or at least atomless BA), and we're able to make the language of BA interpreted in that LTA a CA BA as well, then we have a language that refers to its own sentences, their Boolean combinations, logical equivalence, and truth.

This, in sharp contrast to the setting of Tarski's undefinability of truth: that impossibility result assumes that we have direct access to the syntax of the sentences, represented, e.g., by a Godel number. However in our setting sentences are abstracted, so much so, that they make merely BA elements.

REMARK 7.1. How can we have a theory of BA in which its own LTA makes an atomless BA? One trivial, yet not so useful example, is to take all formulas with unbounded number of free variables. A more userful approach is to add infinitely many uninterpreted constants. Another approach would be to incorporate in the signature infinitely many homomorphism and hemimorphism symbols. It is even easier if the language is extended even further to have a time dimension, cf. remark 8.1.

7.2. The Construction

As hinted above, we shall not merely present a language, but a language-extension mechanism. And as one would already suspect from the previous section, this extension preserves decidability, let alone consistency. We further consider extending many languages at once, and it is indeed yet another feature of our construction to allow languages to co-exist in one unified language, albeit, of course, the interaction between those languages is very limited. However one important way in which those languages can interact is by defining (BA) homomorphisms and hemimorphisms between them, as described in section §5.4, and of course cartesian product (section §5.2). Referring to many BAs at once is easily done by considering the many-sorted theory of BA.

Fix arbitrary languages (the "base logics") in which their formulas (or sentences), up to logical equivalence, make a BA. Then we can consider the many-sorted BA theory interpreted in those BAs. Constants in that languge would be formulas in the base logics. Quantification would take the same semantics of quantification over arbitrary BA elements. If the base logics make an atomless BA, then the extended language has decidable satisfiability iff the base logics have. Otherwise decidable model counting is required, or more precisely, when seen as a BA, to tell whether an element is a disjunction of at least n distinct atoms (as can be seen from section §3.2).

Denote the extended language by $NSO[\mathcal{L}_1,\ldots,\mathcal{L}_n]$ where NSO stands for Nullary Second Order (though not under the usual semantics of nullary relations). We can obtain a language that quantifies over its own formulas (quotiened by logical equivalence) as follows. First, $NSO[\mathcal{L}_1,\ldots,\mathcal{L}_n]$ can already quantify over formulas in $\mathcal{L}_1,\ldots,\mathcal{L}_n$ in the standard fashion of quantification in BA. In this setting, each NSO formula is either true or false, because it is interpreted in a fixed model (being the BA which is the LTA of the base logic), and therfore make a small (only two-element) BA which is typically still far from being elementarily equivalent to the BA of the base logics. To obtain a richer BA from formulas in $NSO[\mathcal{L}_1,\ldots,\mathcal{L}_n]$ we can simply enhance it with infinitely many constant and/or relation and/or function symbols, possibly in a decidability-preserving fashion, e.g. in the ways mentioned in the introduction. Assume $NSO[\mathcal{L}_1,\ldots,\mathcal{L}_n]$ is properly extended such that it now makes an atomless BA (and other kinds of BA are treated similarly). Constants now may be formulas in $NSO[\mathcal{L}_1,\ldots,\mathcal{L}_n]$ appearing inside curly brackets in order to avoid syntactic ambiguity, and handling of quantifiers for the sake of a decision procedure can be done by means of the quantifier elimination decision procedures from chapter 3. The basic syntax of $NSO\left[\mathcal{L}_1,\ldots,\mathcal{L}_n\right]$ (before being extended in any way that makes it an e.g. atomless BA) is therefore

$$\phi := \exists var : sort.\phi|\phi\wedge\phi|\neg\phi|bf=0$$
$$sort := \mathcal{L}_1|\ldots|\mathcal{L}_n|NSO\left[\mathcal{L}_1,\ldots,L_n\right]$$
$$bf := var|\left\{\phi^{sort}\right\}|0|1|bf\cap bf|bf'$$

where $\phi^{\mathcal{L}}$ means any formula in the language $\mathcal{L}$. Each bf may only contain variables and constants from the same sort. The deep-most level of formulas in [nested] curly brakcets will be either a formula in $\mathcal{L}_1,\ldots,\mathcal{L}_n$ or a formula in the language of BA in which the only constants appearing the formula are either 0 or 1. It is then interpreted as a formula over arbitrary atomless BA since they're all elementarily equivalent.

7.3. Splitters

In the setting of NSO, it is commonly desired to calculate a splitter ψ for a formula ϕ. Suppose $\phi(x)$ is a formula in the language of BA and we look for ϕ s.t. $\forall x.\psi(x)\to\phi(x)$ but $\exists x.\psi(x)$ and $\exists x.\phi(x)\wedge\neg\psi(x)$. So ψ puts more strictly constraints on x but still has a satisfying assignment. wlog assume ϕ is quantifier-free. Suppose ϕ is in DNF. If one clause is subsumed by the other, simplify accordingly (we would then say that the DNF is *reduced*). If more than one clause left, then a splitter will choose one of the left clauses, and we're done. So we're left with dealing with splitting a single DNF clause. Assume it is in order normal form as in section 2.3.3. If $a=b$ then no splitter exists (at least not a good splitter, in the case the language is extended to make an atomless LTA). If $a<b$ then it is always possible to choose some constant t which does not equal c_i,d_j, for all i,j. Now add the constraint $x\neq t$.

CHAPTER 8

GSSOTC: A Temporal Logic

8.1. Introduction

In many logical languages it is important to accompany a temporal aspect, in particular in software and process specification languages, to be able to express statements of the form “first do this and then do that”. Moreover, in many cases it is of utmost importance to distinguish between inputs and outputs, and even prove that for any input exists an output, while there can be many inputs and outputs over time, and each output does not depend on future inputs.

Many temporal logics exist from the early days of computer science and computational logic. One of their main limitations is that they commonly turn undecidable once the number of possible states is not finite or not bounded, and further their distinction between inputs and outputs is commonly very limited, if any at all. Overcoming the first limitation is a major active area of research, sometimes referred to as temporal logics over infinite data values. Those data elements are commonly equipped with only very simple operations, typically merely checking for equality. Further, those logics (and other similar machineries e.g. automata) are typically not closed under Boolean combinations.

We devise a new, decidable, family of temporal logics over infinite data values, where those values come with theories much richer than merely equality, in particular with the theory of atomless Boolean Algebras (as well as fixed finite ones though such case does not amount to a significant novelty). Further, this language enjoys the distinctive ability to verify statements of the form “for all inputs exist a well-defined output, possibly depending on the previous output or state”.

Maybe the most notable decidable fragments of first-order logic are the two-variable fragment and the guarded fragment. Boyancyk used the two-variable restriction in order to obtain a decidable temporal logic with infinitely many data values equipped with equality. Here we use a certain guardedness restriction and observe its well-behavedness when it comes to the successor relation.

In what follows there are two kinds of functions involved: one kind of function is simply a way to define sequences. Those are functions from the natural numbers to some BA. The second kind is a function that takes a sequence and returns one. This can be seen as a function that takes inputs, each a BA element, and returns outputs which are again BA elements, this, at each point of time.

REMARK 8.1. We will deal with BAs in general, but the main intended use-case is when those BAs, are LTAs of some other languages to be extended. So this language will speak about sequences (and functions between sequences) of formulas in the underlying languages, and further, about its own formulas, using the very same trick as in NSO, cf. 7.1. Clearly many languages may be enhanced at once in the fashion of many-sorted theory as in NSO.

REMARK 8.2. The construction here can be carried out not only over the theory of atomless BA, but over any weakly-ω-categorical theory. cf. section §5.8. In particular, over the theory of BA interpreted in some finite BA. The latter is very useful to allow low-level computation such as fixed-bitwidth arithmetic, as well as tables as below.

8.2. The Language GS

A formula with ordered free variables (as we shall see) may be seen as defining functions between sequences (and more generally, between trees, in the presence of multiple successor relations), which comes into full play in the language GSSOTC below, but we first describe the language GS dealing with merely sequences and trees and not functions between them. For simplicity we describe the case of sequences while the case for trees is completely analoguous.

A formula with two free variables $\phi(x, y)$ may be seen as defining the set of all sequences such that any two consecutive elements x, y satisfy $\phi(x, y)$. More formally, if $S = s_1, s_2, \dots$ is a sequence, then we say that $S \models \phi(x, y)$ iff $\models \bigwedge_{n=1}^{|S|-1} \phi(s_n, s_{n+1})$ where $|S|$ is the (possibly infinite) length of S. Any definition of sequences by means of free variables (in the above fashion), is equivalent to a definition by means of guarded successor predicate (as below), and vice versa (except for addressing constant positions and the end of the string).

The successor relation $s(n, k)$ over the sort of natural numbers simply means that $n = k + 1$. The above formula may therefore be written as $\forall nk.s(n, k) \to \phi(f(n), f(k))$. The symbol f is a function symbol of type $\mathbb{N} \to \mathcal{B}$ where $\mathcal{B}$ is the underlying BA, and where the structure $\mathbb{N}$ is equipped merely with the successor relation. In this formula the successor relation appears as a guard (in the terminology of the guarded fragment of first order logic). We could also write $\forall mnk.\left[s\left(m,k\right)\wedge s\left(k,n\right)\right]\rightarrow\phi\left(f\left(m\right),f\left(n\right),f\left(k\right)\right)$ where ϕ intends to define sequences where each three consecutive elements satisfy a certain formula ϕ. Strictly speaking, this formula is not in the guarded fragment. We therefore define the guarded successor logic as follows: the successor relation may appear only in guards and only positively, and in a way that uniquely determines the distance between each of the variables. In this formulation, each guarded successor formula is equivalent to a formula of the form $\phi\left(f\left(n\right),f\left(n-1\right),f\left(n-2\right),\dots\right)$, and therefore is equivalent to specifying sequences by means of ordered free variables as above.

So far we used only a universal quantifier over the sort of natural numbers. To handle arbitrary quantification we show how to collapse quantifier alternations. Indeed the guarded formula

$$\forall n\exists k.s\left(n,k\right)\wedge\phi\left(f\left(n\right),f\left(k\right)\right)$$

is easily seen (due to the uniquess of successors) to be equivalent to

$$\forall n\forall k.s\left(n,k\right)\rightarrow\phi\left(f\left(n\right),f\left(k\right)\right)$$

under the interpretation of infinite sequences, and an analoguous translation is trivial for finite sequences: an additional unary relation symbol $\sharp$ denoting the last position in the sequence may appear in the guard. Similarly for both finite and infinite sequences, constant positions may be used. An example of using constant positions and end of sequence would be:

$$\forall n\forall k.\left[s\left(5,n\right)\wedge s\left(12,k\right)\wedge\sharp\left(k\right)\right]\rightarrow\phi\left(f\left(n\right),f\left(k\right)\right)$$

8.3. The Language GSSOTC

We now move to quantification over functions. First consider the more intuitive case where we model a program that at each point of time, reads an input and modifies its internal state, as a function of the previous state as well (here we do not distuingish between states and outputs, since the ability to access the previous output/state puts them in the same line). We can model it as $\phi\left(x_{n-1},x_n,y_n\right)$ where x_{n-1} is the previous state, x_n is the current state, and y_n is the current input. We would like to ask whether for all inputs exists a state. In the spirit of GS we write it as

$$\forall f\exists g\forall nk.s\left(n,k\right)\rightarrow\phi\left(g\left(k\right),g\left(n\right),f\left(n\right)\right)$$

where f is a function from the natural numbers that returns the current input, and g returns the current state (or output). The semantics of the second order quantifier alternation $\forall f \exists g$ is not standard: it reads "for all f exists a time-compatible g". Intuitively it means that the current state may not depend on future inputs. Formally it is defined as follows: any $\forall\exists$ can be written as a $\exists\forall$ where the existential quantifier is of higher order (in the spirit of skolemization), so in our case we can write $\forall f \exists g$ as $\exists G \forall f$ where G is a function from functions to functions, and the above formula can be written as

$$\exists_{tc} G \forall f \forall nk.s\left(n,k\right) \rightarrow \phi\left(\left[G\left(f\right)\right]\left(k\right),\left[G\left(f\right)\right]\left(n\right),f\left(n\right)\right)$$

where we put a restriction on G to be time-compatible, denoted by $\exists_{tc}$, which means that whenever f_1, f_2 share a common prefix[1] of length n, then $G\left(f_1\right), G\left(f_2\right)$ share a common prefix of length n.

This can be enhanced with many input and output functions, and further arbitrary quantifier alternation, allowing to express richer statements e.g. of the form "for each keyboard input, exists a memory state, such that for all network input...". One way to give meaning to that, is to note that if x_n is the input at step n and y_n is the output, then we consider an infinite formula of the form

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \forall x_3 \exists y_3 \ldots.$$

while the infinite alternation is due to the time compatibility condition. Richer quantifier alternation is now straight forward: for all keyboard input at time n, exists a memory state at time n, s.t. for all network input at time n, and so on, is of the form

$$\forall x_1 \exists y_1 \forall z_1 \forall x_2 \exists y_2 \forall z_2 \forall x_3 \exists y_3 \forall z_3 \ldots.$$

8.4. Eliminating Function Quantifiers

We now show an algorithm taking a formula in $GSSOTC$ and use the theory of BA enhanced with recurrence relations (as in section §5.8) to determine whether exists some natural number N such that for all $K \geq N$ exists a sequence of length K satisfying the original formula, which in particular will guarantee the existence of an infinite sequence (or functions between infinite sequences as in from inputs to outputs in a time compatible fashion, and so on). The only property of the theory of atomless BA that we use is that it is weakly ω-categorical, so our construction of temporal logic is therefore applicable to any such language.

[1]We may refer to "prefix" indeed as functions $\mathbb{N} \rightarrow \mathcal{D}$ are indeed equivalently sequences.

A formula in $GSSOTC$ is a prefix of time-compatible function quantifiers, followed by a matrix which for simplicity is assumed to be in DNF. Its general form is therefore

$$Q_1 f_1 \dots Q_k f_k . \bigvee_i \bigwedge_j \ell_{ij} \tag{8.4.1}$$

and each literal ℓ_{ij} is either of the form

$$\forall t_1, \dots, t_n . \gamma_{ij} \to \phi_{ij}$$

or of the form

$$\exists t_1, \dots, t_n . \gamma_{ij} \wedge \phi_{ij}$$

and where each Q is either $\forall$ or $\exists$, γ_{ij} is a conjunction of atoms over $s, \sharp$ in a way that uniquely determines the relative positions as above, and ϕ_{ij} is any formula in the languge $\mathcal{L}$ that may depend on unary $f_1, \dots, f_k$ applied to $t_1, \dots, t_n$ and are therefore of type $\mathbb{N} \to \mathcal{B}$. Note that the function quantifiers must appear as the outermost quantifiers. The universal case is treated here, and the existential case follows from the next section.

To decide satisfiability of the above general form we produce an equisatisfiable formula in BA with recurrence relations. We do so by writing down a recurrence relation indexed by n that expresses "exists a string of length n (or functions between such strings) satisfying the formula", and with free variables denoting the first elements of the sequence (those are to be quantified later on in order to get a final answer). It is a recurrence relation because existence of string of length n can be expressed recursively from a formula expressing existence of string of length $n-1$, and this observation is the key. In the simplest case it may take the form

$$\phi_n(x) := \exists y . \phi(x, y) \wedge \phi_{n-1}(y)$$

where x denotes the first position in the sequence, among other possible forms. Similarly with inputs, suppose x is input and y is output, and given a formula $\phi(x_n, x_{n-1}, y_n, y_{n-1})$, the recurrence relation may take the form

$$\phi_n(x_1, y_1) := \forall x_2 \exists y_2 . \phi_{n-1}(x_2, y_2) \wedge \phi(x_2, x_1, y_2, y_1)$$

It should be noted that if constant positions appear in the formula, e.g. $s(5, n)$, then the number of free variables in ϕ_n cannot be smaller than the largest constant, as those free variables denote the beginning of the sequence. Enforcing the conditions on the beginning of the sequence from the original formula is then done by conjuncting such ϕ_n with those conditions and then quantifying the free variables.

Formulas may be interpreted as speaking about finite strings or about infinite strings. If $\sharp$ is not used, then an infinite string exists iff exists a finite string of any large enough length, which is trivial to check given the method here. If $\sharp$ is used, then since the methods here (including in the next section) can be used to determine logical consequence, we can check whether for the formula entails that strings must be finite, namely whether it entails the formula $\exists n\forall k.s\left(n,k\right)\rightarrow\bot$. Moreover, we can conjunct the original formula with the negation of the latter formula (as below), enforcing infinite strings only.

Remark 8.3. In light of 7.1, we can make GSSOTC speak of sequences of its own sentences, and making it an atomless BA is even easier than NSO: we simply assume a signature with an infinite supply of the above function symbols, i.e. input/output streams.

8.5. Boolean Combination of Sets of Models

Given two formulas e.g. of the form

$$\exists f\forall nk.s\left(n,k\right)\rightarrow\phi\left(f\left(n\right),f\left(k\right)\right)$$

$$\exists f\forall nk.s\left(n,k\right)\rightarrow\psi\left(f\left(n\right),f\left(k\right)\right)$$

their disjunction is simply

$$\exists f\left(\forall nk.s\left(n,k\right)\rightarrow\phi\left(f\left(n\right),f\left(k\right)\right)\bigvee\forall nk.s\left(n,k\right)\rightarrow\psi\left(f\left(n\right),f\left(k\right)\right)\right)$$

and clearly isn't

$$\exists f\forall nk.s\left(n,k\right)\rightarrow\left[\phi\left(f\left(n\right),f\left(k\right)\right)\vee\psi\left(f\left(n\right),f\left(k\right)\right)\right]$$

where the latter expresses a condition of the form "exists a string such that in any position, either ϕ holds or ψ holds". However sometimes we'd like to express a Boolean combination of sets of models (strings and functions between them) like the former formula, and ask whether this Boolean combination is empty. Intersection happens to correspond to conjunction of formulas, but this is not the case for negation and disjunction as we just saw.

Deciding emptiness of a Boolean combination of sets of models (expressed each as a formula) is done as follows. Assume it is given in DNF. Then intersection of positive literals collapses into a single one (i.e. one expressed by a single formula). Emptiness of this DNF can be done by checking whether each single DNF clause is empty. We therefore describe how to decide emptiness of a combination of the form $\left\{\phi\right\}\cap\bigcap_{i=1}^{N}\left\{\psi_i\right\}^C$ where $\left\{\cdot\right\}$ interprets a formula as a set of models (not to be confused with $\left\{\cdot\right\}$ used in NSO), and ^{C} means complementation (negation). This too is done by conversion to BA with recurrence relations. We define η_n^S for each $S \subseteq \left\{1, \ldots, N\right\}$ and is a recurrence relation indexed by n which expresses "there exists a string of length n s.t. ϕ holds in each position, and for all $i \in S$, ψ_i fails in some position". The free variables in η_n^S are typically the values at the initial positions (all in the above fashion), and possibly N more variables which are constrained to be either 0 or 1 and by that not require η_n^S but merely η_n as those variables encode S. Emptiness is then decided by examining satisfiability of $\eta_n^{\{1,\ldots,N\}}$. In its simplest form and in case of one input and one output function symbols with a single input and output lookback, η looks like ([**pp**])

$$\eta_n^S\left(x,y\right)=\forall x'\exists y'.\phi\left(x,y,x',y'\right)\wedge\bigvee_{A\in 2^S}\left[\eta_{n-1}^{S\setminus A}\left(x',y'\right)\wedge\bigwedge_{i\in A}\neg\psi_i\left(x,y,x',y'\right)\right]$$

where the $\bigwedge_{i\in A}\neg\psi_i\left(x,y,x',y'\right)$ part says that the relevant ψ's fail at the initial position, and the part $\eta_{n-1}^{S\setminus A}\left(x',y'\right)$ says that they fail in some other position.

8.6. Execution

Once a GSSOTC formula is satisfiable, and seen as a software specification, we'd like to run that software, which means that at each point of time we receive inputs and calculate outputs. We proceed trivially as follows: we receive inputs and write down a formula with free variables being the current outputs, expressing "there exists a model starting with this prefix", where the prefix is what was already received and calculated until that point of time. Then we find a satisfying subsitution for those free variables (which is guaranteed to exist since the original formula is satisfiable), we output those substitutions, and repeat.

8.7. Revision and Forcing

Consider the following example: a program that gathers knowledge from the user (as inputs), and adds it to some internal knowledgebase. Sometimes the user might input a piece of knowledge which is inconsistent with previous knowledge. But in GSSOTC the look-back is bounded so there's no way to express looking back at all past inputs and take a decision based on them. A remedy for that is introduced by extending the language with what we call a "forcing operator".

First, let's draft the program that adds knowledge to a knowledge-base:

$$\phi\left(x_{n-1},x_n,y_n\right):=\left(x_n=x_{n-1}\wedge y_n\right)$$

where x_n is the current kb, x_{n-1} is the previous one, and y_n is the current input. We would like to force the kb to always be consistent, so we write it as:

$$\phi\left(x_{n-1},x_n,y_n\right):=\left(x_n=x_{n-1}\wedge y_n\right)\wedge F\left(x_n\neq 0\right)$$

which means that in execution time, an extra-logical engine will do some arbitrary operation (e.g. displaying to the user all previous inputs and ask the user to edit them such that the condition $x_n \neq 0$ is satisfied). In consistency check of this GSSOTC formula, we simply assume that $x_n \neq 0$ as we rely on the extra-logical operation (whatever it may be) to force this condition to hold.

In a more general sense, a formula containing the forcing operator F

$$\phi\left(x_{n-1},x_n,y_n\right)=\psi\left(x_{n-1},x_n,y_n\right)\wedge F\left(\chi\left(x_{n-1},x_n,y_n\right)\right)$$

is checked for consistency by reducing it to checking consistency of

$$\forall y_n\exists x_n.\chi\left(x_{n-1},x_n,y_n\right)\rightarrow\psi\left(x_{n-1},x_n,y_n\right)$$

$$\exists y_n\exists x_n.\psi\left(x_{n-1},x_n,y_n\right)\wedge\chi\left(x_{n-1},x_n,y_n\right)$$

8.8. Embedded Execution

In combination with the NSO setting, GSSOTC may take as inputs and outputs sentences in GSSOTC itself. Sometimes we would like to take a GSSOTC program as an input and execute it, e.g. of the form

$$\phi\left(x_n\right):=E\left(x_n\right)$$

which means "execute the input x_n". However clearly execution itself is not expressible in GSSOTC and needs to be an extra-logical operation. We therefore assume that E returns either 0 or 1, depending on whether the execution was successful (note that x_n might look back at the states already computed by ϕ, so consistency is relative to the history of the higher-level program ϕ, its inputs and states). Consistency check is now straight forward: simply rewrite the formula such that the return value of E is either 0 or 1, and check for consistency as usual. An extra-logical engine will perform the execution indeed.

8.9. Splitters

Assume ϕ is a general formula in GSSOTC, so it is a Boolean combination of models, and we'd like to find a splitter ψ, just as in the fashion of (7.3). If that Boolean combination is again a reduced DNF with more than one clause, then simply choose one clause. In case it is a simple clause, there are several ways to go.

During the recurrence relations we always keep the first position a free variable. It is possible to put more constraints on that first position, e.g. in the method presented in (7.3), and that'd be a splitter indeed. In the same fashoin it is possible to put more constraints on any constant positions. This means that we omit possibilities from certain positions.

Another way is to conjunct ψ with the negation of another GSSOTC formula. This means that we omit whole sequences (or functions between them). It is enough to find a single sequence (or function from input to output) and rule out only that sequence (or function).

The case where no good splitter exists can be determined with "exists unique" quantification. Bad splitter always exists: simply add more input or output streams.

We defined a splitter as finding smaller yet nonzero elements. Sometimes it is needed to find bigger yet non-one elements. This is much easier: it can be achieved not only by manipulating the Boolean combination by adding disjuncts or removing conjuncts, but also adding disjuncts or removing conjuncts to the underlying BA formulas themselves.

8.10. Squaring

CHAPTER 9

The Tau 1.0 Language

9.1. Overview

We are now ready to define a language that contains all the extensions in this monograph, which is the Tau language. There is no one Tau language: it depends on which base logics we extend. It therefore consists of the following:

(1) Take GSSOTC over the BAs being:
   (a) The base logics,
   (b) Tau formulas themselves (with models being time-compatible functions between sequences),
   (c) NSO formulas over the base logics,
   (d) The above logics with one free variable s.t. their quantifier is simple,
   (e) All BFs, SBFs, and their higher order counterparts, in those BAs.

(2) In both the GSSOTC level and the NSO level, support:
   (a) Cartesian product,
   (b) Relations with converse,
   (c) Simple quantifiers,
   (d) Infinitely many homomorphism and hemimorphism symbols in the signature,
   (e) Infinitary operations as described,
   (f) Recurrence relations.

(3) The most important base BAs are:
   (a) All finite BAs, encoded as integers wrt bitwise operations, and with addition implemented logically[1],
   (b) All finite BAs of order 2^{2^n} encoded as SBFs of finitely many variables, while syntactically supporting substitution and composition,
   (c) The countable atomless BA SBF,
   (d) Their higher-order counterparts.

[1]Multiplication can also be implemented but will be of very high complexity unless we multiply only by constants, both are easy to implement using recurrence relations.

9.2. Tables

The basic idea is to support functions $2^n \to B$ where B is any BA supported in the Tau language (including products algebra of algebras thereof). This encodes a set of tuples (in the case of product, or 1-tuple if no product is used) where each tuple has an n-bit identifier (possibly taken from prefix codes). Since in this formulation all keys have a value, we set the default value to be zero. It is easy to see how to directly implement this in the Tau language, however we're interested in fixing some syntactic sugar that'll give rise to implementation optimizations. The first kind of atomic formula is of the form

$$T_1 = \text{set}\left(T_2, k, v\right)$$

which means that table T_1 is simply the table T_2 where the value in key k is set to v, overriding any previous value. It is a conservative extension because it can be expressed as

$$T_1\left(k_1, \ldots, k_n\right) = v \wedge$$

$$\forall x_1, \ldots x_n. \left[\bigwedge_i \left(x_i = 0 \vee x_i = 1\right) \wedge x_i \neq k_i\right] \to T_1\left(x_1, \ldots, x_n\right) = T_2\left(x_1, \ldots, x_n\right)$$

where T_1, T_2 are of type BF.

An even more succinct representation is where the table is of the form $2^{2^n} \to B$ so the key $k : SBF\left[n\right]$ is an SBF with n variables. The above atomic formula could then be expressed as

$$T_1\left(k\right) = v \wedge \forall x.x \neq k \to T_1\left(x\right) = T_2\left(x\right)$$

For another kind of atomic formula:

$$T_1 = \text{select}\left(T_2, \phi\left(v\right)\right)$$

which means that T_1 contains all values v in T_2 that satisfy the formula $\phi\left(v\right)$. This is again easily expressed as

$$\forall k \forall v. \left(v = T_2\left(k\right)\right) \to \left(\phi\left(v\right) ? T_1\left(k\right) = v : T_1\left(k\right) = 0\right)$$

Another atomic formula would be

$$u = \bigcap_{v|\phi(v)} T$$

which is an abuse of notation, and intended to mean: take all values v in T that satisfy $\phi\left(v\right)$ and equate their conjunction in u. To this end we first use *select*, and then we're left with computing $u = \bigcap_v T$ which can be expressed as

$$u = \bigcap_{k \in 2^n} T\left(k\right)$$

however to avoid a formula of exponential length we can write a recurrence relation

$$f_n(T) = f_{n-1}(T|_{k_n} = 0) \cap f_{n-1}(T|_{k_n} = 1)$$

and if the implementation allows the user to specify that certain recurrence relations will be unfolded only during runtime, it is easy to see that in many cases, the execution of that recurrence relation will not take exponential time. Clearly recurrence relations will have to be extended to also iterate over a fixed span of argument identifier.

Next we move to intersection and symmetric difference of tables seen as set of tuples. For intersection:

$$T_1 = \text{common}(T_2, T_3)$$

we can express as

$$\forall k \forall v. (T_1(k) = v \wedge T_2(k) = v) ? T_1(k) = v : T_1(k) = 0$$

and similarly for symmetric difference. Next we move to pointwise Boolean operations in tables. This is readily implemented by simple $T_1 = T_2 \cap T_3$ etc. There isn't even a need for quantification over keys as this coincides with the usual Boolean operations over BFs.

For internal optimization, we convert the formula to implicational form where those new atomic formulas are the only ones in the implicants. This can be done in CNF and BDD forms. When the condition is triggered, an internal table modification is performed.

CHAPTER 10

The Two-Variable Fragment with Counting

Pratt-Hartmann has shown how to reduce the satisfiability problem of the two-variable fragment with counting C_2 to an integer linear programming problem. Here we present his complete algorithm to a special case (that still covers the full fragment) but with slightly modified terminology and proof. This work was made mainly by [**pp**].

We are interested in determining whether

$$f\left(C_1,\ldots,C_n,H_1,\ldots,H_m,M_1,\ldots,M_m\right)=0$$

has a solution, where f is a BF, $H_i=M_i^-$, and each H_i is a functional relation, and C is a unary relation (so a cartesian product is involved here). More explicitly, we want a set of minterms to each equal zero:

$$\left(C^{A_i}\times C^{B_i}\right)H^{U_i}M^{V_i}=0$$

We refer to C^{A_i} as the domain of the minterm, and to C^{B_i} as the range of the minterm. Note the abuse of notation here: in the last equation, C,H,M are tuples each. We follow a special case of the algorithm by Pratt-Hartmann and reduce it to an integer linear programming problem. It is a special case because of two points: he considered a more general counting part, but we count here only up to 1. The second point is that non-functional relations don't appear here as we can eliminate them using the above method.

A star-type is a set of minterms in which each H_i appears positively exactly once. The strategy is to find a set of star-types s.t. all minterms in them are nonzero. The idea behind star-types is as follows: suppose $y=h_i\left(x\right)$. Then the pair $\left(x,y\right)$ appears in precisely one minterm (as all minterms are disjoint). Now for each j exists [unique] z s.t. $z=h_j\left(x\right)$. If $x=z$ then H_i,H_j should appear positively at the same minterm, otherwise on different minterms.

The number of star-types with k minterms and with fixed domain, is

$$s_{A_i}\left(n,m,k\right)=2^n2^m2^{k-1}s_{A_i}\left(n,m-1,k-1\right)+k2^ks_{A_i}\left(n,m-1,k\right)$$

because if we add a new minterm to incorporate a new functoinal relation, there are 2^n possible unary parts, 2^m ways to write the converses in the new minterm, and 2^{k-1} to incorporate the new functional relation converse into the existing minterms. If we don't add a new minterm, we have k choices to where to add it positively, and 2^k ways to append its converse. Now

$$s_{A_i}(n,m)=\sum_{k=1}^{m}s(n,m,k)=n^2 2^m\sum_{k=1}^{m}2^{k-1}s_{A_i}(n,m-1,k-1)+\sum_{k=1}^{m}k2^k s_{A_i}(n,m-1,k)$$

and $s(n,m)=2^n s_{A_i}(n,m)$.

$$s_{A_i}(n,m)<2^{n(m+1)}2^{m^2}B_m<2^{m^2}$$

First we treat the problem under the setting of chromatic Z-differentiated structure, where $Z=3m$. This is done by [virtually] assuming that there are ... more unary relations.

Now we need to find a set of nonempty star-types (so all minterms in them are nonempty), that do not contain minterms that are required to be empty by f. Further they have to satisfy conditions on the cardinality of minterms, where each star-type is counted as with fixed domain:

(1) The cardinality of a minterm is the sum of cardinalities of all star-types containing that minterm.
(2) The cardinality of a minterm does not change by taking the converse. [C1]
(3) The cardinality of each minterm in unary relation is either $0,1,>3m$. Each unary relation (or minterms thereof) is the sum of all minterms containing it. [C3]
(4) No star types contains a zero unary minterm. [C2]
(5) If the domain in some minterm is of size 1, then among all minterms containing certain functional relation positively and containing this domain, all are empty except exactly one that has size 1. Moreover, if one star-type contains the same range more than once, then the cardinality of the range is at least the number of minterms with that range in that star-type. [C2]
(6) For each star-type, the number of minterms in it with domain C and range D where $|C|=1$, and all M's appear negatively in it, is no greater than the number of star-types with domain D and no minterm in them has range C. [C4].
(7) If f implies $(C\times D)\,H^0M^0=0$, then in the previous condition, replace "no greater than" with "equals", even for the case $|C|>1$. [C6]
(8) If f implies $(C\times D)\,H^0M^0=0$, then $|C|\le 1$ or $|D|\le 1$. [C5]

DEFINITION 10.1. A *relational minterm* is an expression of the form

$$(\mathcal{U}^A \times \mathcal{U}^B)\,\mathcal{H}^C \mathcal{M}^D$$

where A, B are bitstrings of length n and C, D are of length m. If $C \neq 0$ (namely the bitstring contains at least one 1) then the minterm is called *functional.* If $C \neq 0 \wedge D \neq 0$ then it is *bi-functional.* The domain and range of the minterm, respectively, are U^A, U^B, and by abuse of terminology, we sometimes refer to them as simply A, B. The set of functional minterms will be denoted by $\mathcal{T}_f$, and of bi-functional minterms by $\mathcal{T}_b$. We denote by $\mathcal{T}_0$ the set of all minterms where all H, M appear negatively. For a minterm T we will use T^A, T^B, T^C, T^D to refer to its distinct components.

In this chapter we shall refer to relational minterms as merely minterms. Each minterm is a binary relation over some domain. So we can write $(a, b) \in T$.

DEFINITION 10.2. A *relational counting problem* Z is a set of minterms which is closed under converse, under the following constraints:

(1) The cardinality of all minterms in Z is zero.
(2) Each H_i is a functional relation, and $H_i^- = M_i$.

DEFINITION 10.3. A *star-type* S is a set of functional minterms s.t.

- (S1) All minterms have the same domain,
- (S2) each H occurs positively exactly once in S,
- (S3) if $T \in S$ and $T \in \mathcal{T}_b$ then $T^A \neq T^B$,
- (S4) if $T_1, T_2 \in S$ and $T_1, T_2 \in \mathcal{T}_b$ then $T_1^B \neq T_2^B$.

The last two condtions are called the *chromaticity* conditions. In every model, define $[S] = \bigcap_{T \in S} \pi_1(T)$. The *range* of a star-type is the union of the ranges of all minterms in it.

DEFINITION 10.4. A *solution* to a relational counting problem Z is a set $\mathcal{S}$ of star-types s.t. no star-type contains a minterm from Z, together with a positive extended integer $x_S \in \mathbb{N} \cup \{\infty\}$ assigned to each star-type, under the intention that in a model, $x_S = |[S]|$, and

•

$$\forall T \in \mathcal{T}_b. \sum_{S \mid T \in S} x_S = \sum_{S \mid T^- \in S} x_S \tag{10.0.1}$$

.

•

$$\forall A. \mathcal{S}_A = \emptyset \rightarrow \forall S \in \mathcal{S}. A \nsubseteq \operatorname{ran}(S) \tag{10.0.2}$$

- $$\forall A.\mathcal{S}_A \subset \mathcal{S}_1 \to \forall S \in \mathcal{S}\exists^{\leq 1}T \in S.T^B = A \tag{10.0.3}$$

- $$\forall A.\sum_{S\in\mathcal{S}_A} x_S \leq 1 \vee \sum_{S\in\mathcal{S}_A} x_S \geq 2m+1 \tag{10.0.4}$$

- $$\forall AB\forall S \in \mathcal{S}_1\cap\mathcal{S}_A.\sum_{S^1\in\mathcal{S}_B|A\notin\mathrm{ran}S^1} x_{S^1} \geq \left|\left\{T \in S\backslash\mathcal{T}_b|T^B = B\right\}\right| \tag{10.0.5}$$

- $$\forall T \in Z \cap \mathcal{T}_0.\mathcal{S}_{T^A} \subset \mathcal{S}_1 \vee \mathcal{S}_{T^B} \subset \mathcal{S}_1 \tag{10.0.6}$$

- $$\forall T \in Z\cap\mathcal{T}_0.\mathcal{S}_{T^A} \subset \mathcal{S}_1 \to \sum_{S^1\in\mathcal{S}_{T^B}|T^A\notin\mathrm{ran}S^1} x_{S^1} = \left|\left\{T_1 \in S\backslash\mathcal{T}_b|T_1^B = T^B\right\}\right| \tag{10.0.7}$$

Where the set of all star-types in $\mathcal{S}$ with domain A is denoted by $\mathcal{S}_A$, and $\mathcal{S}_1$ is the set of star-types $S \in \mathcal{S}$ with $x_S = 1$, s.t.

$$S \in \mathcal{S}_1 \cap \mathcal{S}_A \to |\mathcal{S}_A| = 1$$

or in words, if the domain of a certain star-type is a singleton, then there is only one star-type in $\mathcal{S}$ with that domain.

We will show that every such solution is a model and vice versa.

Proposition 10.1. *For any unary relation U, any functional relation H, and any binary relation R,*

$$|U| = 1 \to |(U \times 1)\, HR| \leq 1$$

Proof. Otherwise the range of a singleton element by a function will not be a singleton. □

Corollary 10.1. *If $(a,b) \in H_1R_1$ and $(a,c) \in H_1R_2$ then $b = c$.*

Proposition 10.2. *If T_1, T_2 are distinct minterms and both contain H_i positively then $\pi_1(T_1)\,\pi_1(T_2) = 0$.*

Proof. Suppose $a \in \pi_1(T_1)\,\pi_1(T_2)$. Then $(a, h_i(a)) \in T_1T_2$, but $T_1T_2 = 0$. □

Lemma 10.1. *In every chromatic model of Z, for every domain element a, there is unique S s.t. $a \in [S]$.*

PROOF. Since $\forall a \forall i \exists ! T_i . (a, h_i(a)) \in T_i$ (because each pair is contained in exactly one minterm), put $S = \bigcup_i \{T_i\}$. Each T_i must be functional as otherwise it doesn't contain any $h_i(a)$. If $T_i \neq T_j$ and $T_i, T_j \in S$ then T_j does not contain H_i positively by the previous proposition. So S satisfies S1,S2, while S3,S4 follow from the chromaticity assumption. For uniquess, if $a \in [S_1] \cap [S_2]$ then it belongs to the domain of all minterms in both star-types, and suppose minterm T is in S_1 and not in S_2. Say T contains H_i positively. But in S_2 there is another minterm, different than T, containing H_i positively, which is a contradiction by the previous proposition. □

PROPOSITION 10.3. $\forall T \in \mathcal{T}_f . |T| = |\pi_1(T)|$.

PROOF. $|T| \geq |\pi_1(T)|$ because a projection may never be larger, and $|T| \leq |\pi_1(T)|$ is immediate from the functionality assumption. □

COROLLARY 10.2. *In every model, for every $T \in \mathcal{T}_f$,*

$$|T| = \sum_{S | T \in S} x_S$$

PROOF. Since every domain element belongs to the domain of a unique star-type, so it belongs to the domain of all minterms in that star-type, so $\sum_{S|T\in S} x_S = |\pi_1(T)|$, and use the previous proposition. □

PROPOSITION 10.4. *If a nonzero functional minterm has a singleton domain, then that minterm has a single pair.*

PROPOSITION 10.5. *If two different nonempty minterms have the same singleton range, then their domains are disjoint.*

PROOF. Otherwise they share a common pair. □

PROPOSITION 10.6. *If*

$$A \times B \subseteq \bigcup_{i=1}^{m} H_i \cup M_i$$

then

$$|A \times B| \leq m(|A| + |B|)$$

PROOF. Clearly $|(A \times B) H_i| \leq |A|$ and $|(A \times B) M_i| \leq |B|$, so

$$\left| (A \times B) \bigcup_{i=1}^{m} H_i \cup M_i \right| \leq \sum_{i=1}^{m} |(A \times B)(H_i \cup M_i)|$$

$$\leq \sum_{i=1}^{m} |(A \times B) H_i| + |(A \times B) M_i| \leq m(|A| + |B|)$$

□

THEOREM 10.1. *Existence of chromatic Z-differentiated model implies C1-C6.*

- (10.0.1) is immediate from the fact that converse preserves cardinality, and that the cardinality of a minterm is the sum of cardinalities of all star types containing it, by the previous corollary.
- (10.0.4) is immediate from the Z-differentiated assumption.
- For (10.0.2), if there is no star-type with a certain domain, then that domain is empty (because each H_i is a total function, alternatively because any domain element belongs to some star-type), so it cannot appear as range of non-empty star-type (again by totality).
- For (10.0.3), note that the domain of any minterm in any star-type in $\mathcal{S}_1$, is a singleton. Now use 10.5.
- For (10.0.5), if $U^A = \{a\}$ and $a \in [S]$, consider the set $X = \left\{b \in U^B \middle| (a,b) \in T \in S \backslash \mathcal{T}_b\right\}$ and using 10.4 and the disjointness of minterms, then the cardinality of X is the same as the cardinality of $\left\{T \in S \backslash \mathcal{T}_b | T^B = B\right\}$. We're left with showing that
$$\sum_{S^1 \in \mathcal{S}_B | A \notin \text{ran} S^1} x_{S^1} \geq |X|$$
and for this we show that
$$X \subseteq \bigcup_{S^1 \in \mathcal{S}_B | U^A \notin \text{ran} S^1} \text{dom} S^1$$
Obviously $X \subseteq \bigcup_{S^1 \in \mathcal{S}_B} \text{dom} S^1$ but no element in X has range U^A because (a,b) belongs to a functional minterm which is not bi-functional, so it cannt belong to any bi-functional minterm (by distjointness of minterms).
- For (10.0.6), if $T \in Z \wedge T^C = T^D = 0$ then all elements in T^A are connected to all elements in T^B by at least one function or inverse. If the cardinalities of T^A and T^B are both greater than one, so by the Z-differentiated assumption,
$$\left|T^A \times T^B\right| = \frac{1}{2}\left(\left|T^A\right|\left|T^B\right| + \left|T^A\right|\left|T^B\right|\right)$$
$$\geq \frac{1}{2}\left[\left|T^A\right|(2m+1) + \left|T^B\right|(2m+1)\right] > m\left(\left|T^A\right| + \left|T^B\right|\right)$$
but by 10.6 we should have
$$\left|T^A \times T^B\right| \leq m\left(\left|T^A\right| + \left|T^B\right|\right)$$

- For 10.0.7, in the setting of proving necessity of equation (10.0.5), and further assuming $\left(U^A \times U^B\right)\mathcal{H}^0\mathcal{M}^0 \in Z$, we want to show that
$$X = \bigcup_{S^1 \in \mathcal{S}_B | U^A \not\subseteq \mathrm{ran} S^1} \mathrm{dom} S^1$$
One side is already proved above. For the other direction, if $b \in \bigcup_{S^1 \in \mathcal{S}_B | U^A \not\subseteq \mathrm{ran} S^1} \mathrm{dom} S^1$ then it has a star-type S^1 and b is not in its range, so (b, a) cannot belong to any functional minterm in S^1, so (a, b) is either in a functional minterm, or in $\left(U^A \times U^B\right)\mathcal{H}^0\mathcal{M}^0$. The latter possibility is forbidden by assumption. So $b \in X$ by definition of X.

The model:

(1) The domain D is of size $\sum_S x_S$. Arbitrarily and for every S, assign the star-type S to x_S elements, uniquely. Write $a \in S$ if a is assigned a star-type S. We assume that the domain is a subset of the natural numbers, as we will need a linear order among the domain elements.
(2) Each $x \in D$ of star-type S is a member of dom$S$. This defines the unary relations in the model.
(3) The set of all elements that are assigned a star-type that contains the minterm T will be denoted by D_T.
(4) For each A where $\left|\mathcal{U}^A\right| \geq 2m + 1$ fix P_1^A, P_2^A where $\left|P_1^A\right| \geq m \wedge \left|P_2^A\right| \geq m \wedge P_1^A \cap P_2^A = \emptyset \wedge P_1^A \cup P_2^A = \mathcal{U}^A$.
(5) Assume unary minterms are linearly ordered, so we can write e.g. $T^A \leq T^B$ which would mean e.g. that $A \leq B$ where the bitstrings A, B are considered as numbers.
(6) For each nonzero minterm $T \in \mathcal{T}_b$ (the ones that appear in nonzero star-types) with $T^A \leq T^B$ (in order to avoid treating T, T^- in conflicting ways), clearly $|D_T| = |D_{T^-}|$ by 10.0.1. So there is a bijection between D_T, D_{T^-}. Let T be precisely such a bijection (and correspondingly for T^-). Note that $T^A \neq T^B$ by the chromaticity assumption (so in particular the bijection has no fixed-points), and similarly no pair is chosen to belong to two different minterms, and also that $T \neq T^-$.
(7) For every $A \neq B$ and for every nonempty S with $A = \mathrm{dom} S$, and every $a \in S$:
    (a) If $\left|\mathcal{U}^A\right| \geq 2m + 1$ and $\left|\mathcal{U}^B\right| \geq 2m + 1$. Consider all minterms $T \in S \backslash \mathcal{T}_b$, with $T^B = \mathcal{U}^B$ and for each such T choose $b_T \in \mathcal{U}^B$ s.t. if $A < B$ and $a \in P_i^A$ then $b_T \in P_i^B$, and if $B < A$ and $a \in P_i^A$ then $b_T \in P_{3-i}^B$, and declare $(a, b_T) \in T$. Similarly set $(b_T, a) \in T^-$. We select b_T s.t.

(a, b_T) was not assigned to any other minterm beforehand (including in the treatment of $\mathcal{T}_b$). This is always possible because there are at most m minterms in the star-type assigned to a so we can always find such an element b in P_1^B, P_2^B.

(b) If $\left|\mathcal{U}^A\right| \geq 2m+1$ and $\left|\mathcal{U}^B\right| = 1$, for any $T \in S \backslash \mathcal{T}_b$ with $T^B = \mathcal{U}^B$ take b_T to be the unique element in $\mathcal{U}^B$ and set $(a, b_T) \in T$. This pair was not chosen before because since $\mathcal{S}_B \subset \mathcal{S}_1$, 10.0.3 implies that

$$\forall S \in \mathcal{S} \exists ! T \in S . T^B = B$$

so in the star-type of a there is only one minterm with range B. Clearly also set $(b_T, a) \in T^-$. Note that T^- is not functional so it doesn't belong to any star-type (similarly in the previous step).

(c) If $\left|\mathcal{U}^A\right| \geq 2m+1$ and $\left|\mathcal{U}^B\right| = 0$, then by equation (10.0.2), S does not contain a minterm with range $\mathcal{U}^B$.

(8) For every $A \neq B$ and for every nonempty S with $A = \mathrm{dom} S$, and every $a \in S$, if $\left|\mathcal{U}^A\right| = 1$, for any $T \in S \backslash \mathcal{T}_b$ with $T^B = \mathcal{U}^B$, take any b s.t. (a, b) was not previously assigned. If $b \in \mathcal{U}^B$ and its star-type is S^b, and $\mathcal{U}^A \nsubseteq \mathrm{ran} S^b$ then such b was not assigned in 6 or in 7. By 10.0.5 the number of such b is at least $\left|\left\{T \in S \backslash \mathcal{T}_b \mid T^B = B\right\}\right|$ which is the number of elements needed.

(9) For all functional minterms T with $T^A = T^B$. Then $\left|T^A\right| > 1$ as otherwise equation (10.0.5) would evaluate to $0 \geq 1$. Then $\left|T^A\right| \geq 2m+1$, call them $t_1, \ldots, t_k$ and recall that they are natural numbers. To each t_i we find

$$j \in \bigcup_{p=0}^{m-1} \{(i+p) \bmod k\}$$

s.t. (t_i, t_{j+1}) is not previously assigned, and declare $(t_i, t_{j+1}) \in T \wedge (t_{j+1}, t_i) \in T^-$. Since $\left|T^A\right| \geq 2m+1$, T, T^- will never be assigned the same pair. We also know that exists such not previously assigned pair because there are at most m minterms in the star-type of t_i, so there are at most $m-1$ previously assigned such pairs. For T^-, none of this pairs was used for T^- because by definition of j, $(t_i, t_{j+1}) \neq (t_{j'+1}, t_{i'})$ for any i', j'.

(10) For $(a, b) \in \mathcal{U}^A \times \mathcal{U}^B$ not yet assigned to any minterm, we assign $\left(\mathcal{U}^A \times \mathcal{U}^B\right) \mathcal{H}^0 \mathcal{M}^0$ and we have to show that either this

minterm does not belong to Z, or such (a, b) don't exist. If that minterm is in Z then by equation (10.0.6) $\left|\mathcal{U}^A\right| = 1 \vee \left|\mathcal{U}^B\right| = 1$. By symmetry it's enough to consider $\left|\mathcal{U}^A\right| = 1$. If S^a is the star-type of a, then by equation (10.0.7)

$$\sum_{S \in \mathcal{S}_{\mathcal{U}^B} | \mathcal{U}^A \nsubseteq \text{ran} S} x_S = \left|\left\{T \in S^a \backslash \mathcal{T}_b | T^B = \mathcal{U}^B\right\}\right|$$

so a minterm is already assigned to (a, b) since, if S^b is the star-type of b and $\mathcal{U}^A \subseteq \text{ran} S^b$, then a is the image of b over some function, so are excluded from above sum in the lhs, while the rhs describes already-assigned functions from a to b in 8. The lhs guarantees that we assigned all such pairs.

LEMMA 10.2. *Ackermann lemma, the unary case: if C appears positively in ψ then*

$$\exists C \forall x. \left[Cx \to \phi(x)\right] \wedge \psi(C) \equiv \psi(C)^{Cx}_{\phi(x)}$$

and if negatively then

$$\exists C \forall x. \left[\phi(x) \to Cx\right] \wedge \psi(C) \equiv \psi(C)^{Cx}_{\phi(x)}$$

note that C does not appear in ϕ.

COROLLARY 10.3. *Assume ψ is in NNF and has a non-atomic subformula of the form $\phi(x)$, then it is equisatisfiable with replacing $\phi(x)$ with Cx and rewrite ψ as $\forall x. Cx \to \phi(x)$ conjuncted with the modified ψ.*

$$\forall x. \phi(x) \vee \forall y. \psi(x, y) \vee \forall z. \chi(y, z)$$

$$\left[\forall xy. Rxy \to \psi(x, y) \vee \forall z. \chi(y, z)\right] \forall x. \phi(x) \vee \forall y. Rxy$$

$$\left[\forall xy. \left(Rxy \to \psi(x, y)\right) \vee \left(Rxy \to \forall z. \chi(y, z)\right)\right] \forall x. \phi(x) \vee \forall y. Rxy$$

PROPOSITION 10.7. *The formula*

$$\forall x. \phi(x) \vee \forall y. \psi(x, y) \vee \forall z. \chi(y, z)$$

is equisatisfiable with

$$\left[\forall xy. Cx \to \chi(x, y)\right] \wedge \forall xy. \phi(x) \vee \psi(x, y) \vee Cy$$

Similarly

$$\exists x. \phi(x) \wedge \exists y. \psi(x, y) \wedge \exists z. \chi(y, z)$$

is equisatisfiable with

$$(\exists! x. Cx) \wedge \left[\exists xy. Cx \wedge \chi(x, y)\right] \wedge \exists xy. \phi(x) \wedge \psi(x, y) \wedge Cy$$

PROPOSITION 10.8. *In the two-variable fragment and in a formula in NNF, a subformula of the form*

$$\forall x.\phi\left(x,t\right)\vee\forall y.\psi\left(x,y\right)\vee\forall z.\chi\left(y,z\right)$$

can be replaced with

$$\left[\forall xy.Cx\rightarrow\chi\left(x,y\right)\right]\wedge\forall xy.\phi\left(x,t\right)\vee\psi\left(x,y\right)\vee Cy$$

while maintaining satisfiability. Similarly

$$\exists x.\phi\left(x,t\right)\wedge\exists y.\psi\left(x,y\right)\wedge\exists z.\chi\left(y,z\right)$$

then if this subformula appears under universal quantifiers, then is equisatisfiable with

$$\left[\forall y.Cy\rightarrow\exists z.\chi\left(y,z\right)\right]\wedge\exists x.\phi\left(x,t\right)\wedge\exists y.\psi\left(x,y\right)\wedge Cy$$

and if not, this formulation is still valid, but can be replaced with the more efficient

$$\left(\exists!x.Cx\right)\wedge\left[\exists xy.Cx\wedge\chi\left(x,y\right)\right]\wedge\exists xy.\phi\left(x\right)\wedge\psi\left(x,y\right)\wedge Cy$$

In all cases, all but the last conjunct can be conjuncted from the outside with the whole formula.

PROPOSITION 10.9. *The formula* $\exists x.\phi\left(x\right)\wedge\forall y.\psi\left(x,y\right)$ *is equisatisfiable with*

$$\left(\exists!x.Cx\right)\wedge\left[\forall xy.Cx\rightarrow\psi\left(x,y\right)\right]\wedge\exists x.\phi\left(x\right)\wedge Cx$$

and similarly for subformulas (assuming NNF) that do not fall under universal quantifiers. Otherwise those subformulas can be replaced with

$$\left[\forall xy.Cx\rightarrow\psi\left(x,y\right)\right]\wedge\exists x.\phi\left(x\right)\wedge Cx$$

and moreover, the first conjunct can be conjuncted from the outside with the whole formula.

PROPOSITION 10.10. *Assuming NNF, the subformula*

$$\forall x.\forall y.\phi\left(x,y\right)\vee\forall y.\psi\left(x,y\right)$$

is equisatisfiable with

$$\left[\forall xy.Cx\rightarrow\psi\left(x,y\right)\right]\wedge\forall xy.\phi\left(x,y\right)\vee Cx$$

similarly

$$\exists x.\exists y.\phi\left(x,y\right)\wedge\exists y.\psi\left(x,y\right)$$

is equisatisfiable with

$$\left[\forall x.Cx\rightarrow\exists y.\psi\left(x,y\right)\right]\wedge\exists xy.\phi\left(x,y\right)\wedge Cx$$

and if the subformula does not fall under a universal quantifier, then we can also write it more efficiently

$$(\exists! x.Cx) \wedge [\exists xy.Cx \wedge \psi(x,y)] \wedge \exists xy.\phi(x,y) \wedge Cx$$

CHAPTER 11

Exercises

Exercise 11.1. Show that $f(f(f(x))) = f(x)$ for any BF f.

Exercise 11.2. Show that $f(x+y+z) = f(x)+f(z)+f(z)$ for any BF f.

Exercise 11.3. Show that $\exists x.f(x) = 0$ iff $f(f(0)) = 0$, and that $\forall x.f(x) = 0$ iff $f(f(1)) = 0$, for any BF f.

Exercise 11.4. Show that an SBF has a zero in some BA iff it has a zero in all BAs.

Exercise 11.5. Show that in atomic BA, a BF has a zero which is an atom iff $f'(1) \neq 0$ and $|f(0)| \leq 1$. Show that all its zeros are atoms iff $f(0) = f'(1)$ and $|f'(1)| = 1$, in which case it has a single zero.

Exercise 11.6. Show that $f(x) = f(0) + x\frac{\partial f}{\partial x}$, for any BF f (this is Davio's decomposition which gives rise to the Reed-Muller decomposition).

Exercise 11.7. There are 2^{2^n} SBFs of n variables. But $2^{2^n} = \left(2^{2^{n-1}}\right)^2$ as well as $2^{2^n} = 1+\prod_{k=1}^{n-1}\left(2^{2^{k-1}}+1\right)$. Prove the last identity. Those two representations of 2^{2^n} hint to two possible decompositions of SBFs. Find out two such matching decompositions.

Exercise 11.8. Let $\mathcal{B}$ be the BA of SBFs with unboundedly many variables, so each SBF depends only on finitely many variables $x_1, x_2, \ldots$, but unboundedly so. Show that $\mathcal{B}$ is an atomless BA.

Exercise 11.9. Let $\mathcal{B}$ be the BA from the previous exercise. Show that any homomorphism $\mathcal{B} \to \mathcal{B}$ can be written as a set of substitutions $x_i \to f_i$ where f_i is an SBF, so any homomorphism takes an SBF and replaces a variable (or several) with SBFs.

Exercise 11.10. Let $\mathcal{B}$ be the BA from the previous exercise. Show that all ultrafilters (namely all homomorphisms into the two-element BA) can be identified with the set of all infinite bitstrings.

Exercise 11.11. Prove that $f(y) \leq x \leq g(y)$ iff $x'f(0)+xg'(0) \leq y \leq x'f'(1)+xg(1)$.

Exercise 11.12. Prove that $x \nleq f(y)$ iff $xf'(0) \nleq y \vee y \nleq x' \cup f(1)$.

Exercise 11.13. Prove that $f(y) \nleq x$ iff $x'f(0) \nleq y \vee y \nleq x \cup f'(1)$.

Exercise 11.14. Show a direct proof that the system

$$ax \neq 0$$

$$bx' \neq 0$$

has a solution iff a, b are not equal atoms.

Exercise 11.15. Prove theorem 4.1.

Exercise 11.16. Prove theorem 4.2.

Exercise 11.17. The system in theorem 4.2 can be used to solve systems in which only SBFs appear, but apparently not BFs. Show that this is not the case, namely show how to convert the general BF case to the theorem's setting.

Exercise 11.18. Prove theorem 4.3.

Exercise 11.19. Prove lemma 4.4.

Exercise 11.20. Prove the correctness of the algorithm appearing after theorem 4.1.

Exercise 11.21. Prove theorem 5.1.

Exercise 11.22. Show that for any BF f, defining a sequence by means of $\phi(x, y) := f(x, y) = 0$ and checking whether a sequence of lengths $2, 3, \ldots$ exists, converges after two iterations. This demonstrates that satisfiability of GSSOTC formulas may take a surprisingly small number of steps.

Bibliography

[pp] Pawel Parys, private communication.
[rud1] Rudeanu, “Boolean functions and equations”
[rud2] Rudeanu, “Lattice functions and equations”
[bro] Brown, “Boolean reasoning”
[mo] Marriot, Odersky
[kun] Kuncak
[rev] Revesz
[tar] Tarski
[koz] Kozen
[ck] Chang & Keisler
[giv] Givant
[hal] Hall
[kop] Koppelberg, “Handbook of Boolean Algebras”.
[hal] Halmos, 1962. Algebraic Logic. New York: Chelsea.

Index