Boolean Logic

Lecture Notes

Gert Smolka
Saarland University
November 29, 2014

1 Introduction

Boolean logic is the most straightforward logic there is. It has important practical applications and a rich structure providing for the exposition of many logical ideas. There are different interpretations of boolean logic. The canonical interpretation is concerned with operations on a two-valued set. The algebraic interpretation relates boolean logic with operations on power sets.

We will focus on semantic equivalence of boolean expressions and study decision procedures and equational proof systems. Our decision procedures will be based on the prime tree normal form used with reduced and ordered BDDs. Completeness and decidability of an equational proof system for boolean algebra will be shown using the results developed for prime trees.

We will show that the equational proof system is sound and complete for every boolean algebra. This result will turn out to be a straightforward consequence of the results for two-valued semantic equivalence.

The foundational basis for our development of boolean logic is constructive type theory. The text is high-level and assumes that the reader is familiar with constructive type theory and the proof assistant Coq. Our definitions and results are presented informally but with constructive type theory and Coq in mind. We expect that the reader carries out in Coq substantial parts of the mathematical development presented in this text.

Our treatment of boolean logic brings together different aspects of computational logic: equational proof systems, unique normal forms, concrete and abstract semantics, and decision procedures. Constructive type theory provides us with a foundation elegantly accommodating all these aspects. Our development fully exploits that constructive type theory gives us a unified framework for algorithms, proofs, and verification of algorithms. Constable [4] emphasizes this

distinguished aspect of constructive type theory by speaking of *computational* type theory.

2 Boolean Operations

We start with a type **bool** consisting of two elements **true** and **false**:

$$\mathbf{B} \;::=\; \top \mid \bot$$

We call the elements of $\mathbf{B}$ **booleans**.

We define operations on $\mathbf{B}$ known as **negation**, **conjunction**, **disjunction**, and **conditional**.

$$\begin{array}{llll} \neg\top := \bot & \top \wedge b := b & \top \vee b := \top & \mathsf{C}\top bc := b \\ \neg\bot := \top & \bot \wedge b := \bot & \bot \vee b := b & \mathsf{C}\bot bc := c \end{array}$$

Conditionals will play an important role in the following. They express a boolean case analysis reminiscent of an if-then-else expression in functional programming languages.

An **identity** is a provable equation. The following identities say that conditional can express negation, conjunction, and disjunction.

$$\neg a = \mathsf{C}a\bot\top \qquad a \wedge b = \mathsf{C}ab\bot \qquad a \vee b = \mathsf{C}a\top b$$

Together, conjunction and negation can express disjunction and conditional.

$$a \vee b = \neg(\neg a \wedge \neg b) \qquad \mathsf{C}abc = a \wedge b \vee \neg a \wedge c$$

Proving the above identities is straightforward: Case analysis on the boolean variables and definitional equality will do the job.

Incidentally, boolean **implication** is defined as follows:

$$\begin{aligned} \top \to b &:= b \\ \bot \to b &:= \top \end{aligned}$$

We have $s \to t = \mathsf{C}st\top$ and $s \to t = \neg s \vee t$.

In Coq one may use the predefined type *bool* for $\mathbf{B}$. The library *Bool* provides negation (*negb*), conjunction (*andb*), conjunction (*orb*), conditional (*ifb*), and implication (*implb*).

3 Conditional Expressions

We now define a class of syntactic expressions. We choose to work with conditional as the single operation. This design decision comes without loss of generality since conditional can express the other operations. Expressions with conditionals will be convenient since they can represent decision trees without coding.

We define a type of **conditional expressions** as follows:

$$s, t, u, v \ ::= \ \top \mid \bot \mid x \mid \mathsf{C}stu \qquad\qquad (x \in \mathbf{N})$$

Expressions of the form x are called **variables**. By definition there is a variable for every number. We will call numbers **variables** when we use them to obtain expressions.

Note that we use overloaded notation. For instance, $\bot$ denotes either a boolean or an expression, and x denotes either a number or an expression. Overloading is common in mathematical presentations. The accompanying Coq development does not use overloading.

An **assignment** is a function from variables to booleans. The letter α will be reserved for assignments. Note that an assignment fixes a value for every variable. Given an assignment, we can **evaluate** an expression to a boolean as one would expect:

$$\begin{aligned} eva\ \alpha\ \top &:= \top \\ eva\ \alpha\ \bot &:= \bot \\ eva\ \alpha\ x &:= \alpha x \\ eva\ \alpha\ (Cstu) &:= \mathsf{C}\ (eva\ \alpha\ s)\ (eva\ \alpha\ t)\ (eva\ \alpha\ u) \end{aligned}$$

We will use the notation $\hat{\alpha}s := eva\ \alpha\ s$.

An expression is **satisfiable** if there is an assignment under which it evaluates to true. An expression is **valid** if it evaluates to true under all assignments.

Two expressions are semantically **equivalent** if they evaluate to the same boolean under every assignment:

$$s \approx t \ := \ \forall \alpha.\ \hat{\alpha}s = \hat{\alpha}t$$

Two expression are semantically **separable** if there exists an assignment under which they evaluate to different booleans. We speak of a **separating** assignment.

We have already seen that conditional can express negation, conjunction, and disjunction. Based on the respective identities we define **abbreviation functions** for expressions (note the overloading).

$$\neg s := \mathsf{C}s\bot\top \qquad\qquad s \wedge t := \mathsf{C}st\bot \qquad\qquad s \vee t := \mathsf{C}s\top t$$

Fact 1 $\hat{\alpha}(\neg s) = \neg(\hat{\alpha}s)$, $\hat{\alpha}(s \wedge t) = \hat{\alpha}s \wedge \hat{\alpha}t$, $\hat{\alpha}(s \vee t) = \hat{\alpha}s \vee \hat{\alpha}t$.

Fact 2 $\mathsf{C}stu \approx s \wedge t \vee \neg s \wedge u$, $\neg s \approx \mathsf{C}s\bot\top$, $s \wedge t \approx \mathsf{C}st\bot$, $s \vee t \approx \mathsf{C}s\top t$.

Fact 3 $\neg\neg s \approx s$ and $\neg\mathsf{C}stu \approx \mathsf{C}s(\neg t)(\neg u)$.

Fact 4 (Reductions)

1. s is valid if and only if $\neg s$ is unsatisfiable.
2. s is unsatisfiable if and only if $\neg s$ is valid.
3. $s \approx t$ if and only if $\mathsf{C}st(\neg t)$ is valid.
4. s is valid if and only if $s \approx \top$.
5. s is unsatisfiable if and only if $s \approx \bot$.
6. s and t are separable if and only if $\mathsf{C}s(\neg t)t$ is satisfiable.
7. s is satisfiable if and only if s and $\bot$ are separable.

Fact 5 With excluded middle, the following equivalences can be shown.

1. s is satisfiable if and only if $\neg s$ is not valid.
2. s and t are separable if and only if $s \not\approx t$.

Note that only the directions from right to left need excluded middle. Moreover, it suffices to assume excluded middle for satisfiability. We will prove that satisfiability is computationally decidable. Thus the equivalences of Fact 5 can be shown without assumptions.

Fact 6 Semantic equivalence is an equivalence relation compatible with C:

$$\frac{s \approx s' \qquad t \approx t' \qquad u \approx u'}{\mathsf{C}stu \approx \mathsf{C}s't'u'}$$

The above fact says that semantic equivalence is an abstract equality relation. This fact is useful for proofs. On paper one uses the equivalence properties automatically when one sees an abstract equality notation like $s \approx t$. In Coq one may register semantic equivalence with setoid rewriting so that one can use the equality tactics (i.e., *rewrite*) for semantic equivalence.

Exercise 7 Define the abbreviation functions $\neg s$, $s \wedge t$, and $s \vee t$ in Coq and prove the above facts.

A useful intuition says that two expressions are equivalent if they denote (i.e., describe) the same *boolean function* (Bryant [3], Whitesitt [7]). We can make this intuition precise by taking the function $\lambda\alpha.\hat{\alpha}s$ as the boolean function described by the expression s. This will work in type theory if we assume that functions

are extensional (as in set theory). If instead of infinitely many variables we work with finitely many variables, we can model boolean functions simply as functions $\mathbf{B}^n \to \mathbf{B}$ where n is the number of different variables.

Exercise 8 Design and verify a function σ that for two expressions s and t yields an expression σst such that an assignment α separates s and t if and only if it satisfies σst.

4 Basic Decision Procedure

We now construct an informative decision procedure for satisfiability of expressions that yields a satisfying assignment in the positive case. Given this procedure, the construction of decision procedures for validity, equivalence and separation is straightforward (due to Fact 4).

We base the decision procedure for satisfiability on a variable elimination method sometimes referred to as British Museum method. The idea is to search for a satisfying assignment by replacing the variables in the expression by either $\top$ or $\bot$ and check whether the resulting variable-free instance of the expression evaluates to $\top$.

The principles behind the decision procedure can be formulated with three facts.

Fact 9 (Coincidence) $\hat{\alpha}s = \hat{\beta}s$ if α and β agree on all variables occurring in s.

For the formal statement of the coincidence lemma, we need a function $\mathcal{V}s$ that for an expression s yields a list containing exactly the variables occurring in s (double occurrences are fine). Such a function can be defined by structural recursion on s and will be assumed in the following.

We call an expression **ground** if it contains no variables. The coincidence lemma tells us that it suffices to evaluate ground expressions with the **default assignment** $\alpha_\bot := \lambda x.\bot$. We say that a ground expression **evaluates** to a boolean b if $\widehat{\alpha_\bot}\, s = b$.

A ground expression is valid if and only if it is satisfiable if and only if it evaluates to $\top$. Two ground expressions are semantically equivalent if and only if they evaluate to the same value.

Fact 10 (Ground Satisfiability) A ground expression is satisfiable if and only if it is valid if and only if it evaluates to true.

We assume a **substitution operation** s_t^x that yields the expression obtained from an expression s by replacing every occurrence of the variable x with the expression t.

Fact 11 (Variable Elimination) Let s be an expression and x be a variable. Then s is satisfiable if and only if either $s^x_\top$ or $s^x_\bot$ is satisfiable.

The proof of this fact relies on a correspondence between assignments and substitutions. We use the **update notation** α^x_b to denote the assignment that maps x to b and otherwise agrees with α.

Fact 12 $\hat{\alpha}(s^x_t) = \widehat{\alpha^x_{\hat{\alpha}t}}\, s$.

Fact 13 $s^x_t = s$ if $x \notin \mathcal{V}s$.

We now realize the decision procedure with a function

$$\mathit{check} : \mathit{list}\ \mathbf{N} \to \mathit{exp} \to (\mathbf{N} \to \mathbf{B})_\bot$$

satisfying the specification stated by Theorem 14. Note that $X_\bot$ is notation for the option type for X. We define *check* by structural recursion on the list argument.

$$\begin{aligned}
\mathit{check}\ \mathit{nil}\ s &:= \mathsf{if}\ \widehat{\alpha_\bot}\, s = \top\ \mathsf{then}\ \lfloor \alpha_\bot \rfloor\ \mathsf{else}\ \bot \\
\mathit{check}\ (x :: A)\ s &:= \mathsf{match}\ \mathit{check}\ A\ (s^x_\top),\ \mathit{check}\ A\ (s^x_\bot)\ \mathsf{with} \\
&\quad \mid \lfloor \alpha \rfloor,\ _ \implies \lfloor \alpha^x_\top \rfloor \\
&\quad \mid _,\ \lfloor \alpha \rfloor \implies \lfloor \alpha^x_\bot \rfloor \\
&\quad \mid \bot,\ \bot \implies \bot
\end{aligned}$$

Theorem 14 (Correctness) Let $\mathcal{V}s \subseteq A$. Then:

1. If $\mathit{check}\ A\ s = \lfloor \alpha \rfloor$, then $\hat{\alpha}s = \top$.
2. If $\mathit{check}\ A\ s = \bot$, then s is unsatisfiable.

Corollary 15 Satisfiability, validity, equivalence, and separability of expressions are decidable.

Corollary 16

1. $s \not\approx t$ if and only if s and t are separable.
2. s is not valid if and only if $\neg s$ is satisfiable.

5 Prime Tree Procedure

We now improve the basic decision procedure by making it more informative. Given an expression, the improved procedure will return an equivalent expression in so-called prime tree normal form. The prime tree normal form of a valid

expression is always $\top$, and the prime tree normal form of an unsatisfiable expressions is always $\bot$. Thus an expression is satisfiable if and only if its prime tree normal form is different from $\bot$.

We define **prime expressions** inductively:

1. $\bot$ and $\top$ are prime expressions.
2. $\mathsf{C}xst$ is a prime expressions if s and t are different prime expressions and x is a variable smaller than every variable y occurring in s or t (i.e., $x < y$).

Prime expressions can be depicted as decision trees. In fact, prime expressions are ordered and reduced decision trees as they appear with ordered and reduced binary decision diagrams in Bryant [3]. The word reduced means that the direct subtrees of a tree must be different. To make the connection with decision trees explicit, we refer to prime expressions also as **prime trees** and speak of the **prime tree normal form**.

It is not difficult to modify the basic decision procedure so that it returns equivalent prime trees. The prime tree for an expression mirrors the variable elimination steps the procedure does for the expression. If we start the basic procedure with a strictly sorted list of variables, the order constraint for prime trees will be fulfilled. If in addition when we construct the tree witnessing the elimination of x check whether the two recursively obtained subtrees u and v are identical and return just u if they are and $\mathsf{C}xuv$ otherwise, we will obtain ordered and reduced decision trees. Here are three facts that ensure that the obtained prime tree is equivalent to the given expression.

Fact 17 (Ground Evaluation) $s \approx \hat{\alpha}\, s$ if s is ground.

Fact 18 (Shannon Expansion) $s \approx \mathsf{C}x(s^x_\top)(s^x_\bot)$.

Fact 19 (Reduction) $\mathsf{C}stt \approx t$.

We realize the prime tree procedure with a function η satisfying Theorem 23. We obtain η with a function η' such that $\eta s = \eta' A\, s$ if A is a strictly sorted list[1] of variables containing the variables occurring in s.

$$\begin{aligned}
\eta s &:= \eta'\ (\mathit{sort}\ (\mathcal{V} s))\ s \\
\eta'\ \mathit{nil}\ s &:= \widehat{\alpha_\bot}\ s \\
\eta'\ (x :: A)\ s &:= \mathit{red}\ x\ (\eta' A\ (s^x_\top))\ (\eta' A\ (s^x_\bot)) \\
\mathit{red}\ x\ s\ t &:= \mathsf{if}\ s = t\ \mathsf{then}\ s\ \mathsf{else}\ \mathsf{C}xst
\end{aligned}$$

Lemma 20 $\eta' A\, s \approx s$ if $\mathcal{V} s \subseteq A$.

[1] A list $[x_1, \ldots, x_n]$ is strictly sorted if $x_1 < \cdots < x_n$.

Lemma 21 $\mathcal{V}(\eta' A\, s) \subseteq A$.

Lemma 22 $\eta' A\, s$ is prime if A is strictly sorted.

Theorem 23 (Correctness) ηs is prime, $\eta s \approx s$, and $\mathcal{V}(\eta s) \subseteq \mathcal{V} s$.

Note that the identities $\eta\bot = \bot$, $\eta\top = \top$, $\eta x = \mathsf{C}x\top\bot$, and $\eta(\neg x) = \mathsf{C}x\bot\top$ hold by definitional equality.

We now show that the prime tree normal form is unique.

Theorem 24 (Separation) Different prime trees are separable.

Proof Let s and t be prime trees. We construct a separating assignment by nested induction on the primeness of s and t. Case analysis.

1. The prime trees $\top$ and $\bot$ are separated by every assignment.
2. Consider two prime trees $\mathsf{C}xst$ and u where x does not occur in u. We have either $s \neq u$ or $t \neq u$. We assume $s \neq u$ without loss of generality. The inductive hypothesis gives us an assignment α separating s and u. The claim follows since $\alpha^x_\top$ separates $\mathsf{C}xst$ and u.
3. Consider two different prime trees $\mathsf{C}xst$ and $\mathsf{C}xuv$. We have either $s \neq u$ or $t \neq v$. We assume $s \neq u$ without loss of generality. The inductive hypothesis gives us an assignment α separating s and u. The claim follows since $\alpha^x_\top$ separates $\mathsf{C}xst$ and $\mathsf{C}xuv$. ∎

Our proof of the separation theorem omits details that must be filled in a formal development. The idea is that the reader can fill in the details if he tries hard enough. Some of the important insights are made precise by the following lemma.

Lemma 25

1. $\alpha^x_\top$ separates $\mathsf{C}xst$ and u if α separates s and u and $x \notin \mathcal{V}s + \mathcal{V}u$.
2. $\alpha^x_\bot$ separates $\mathsf{C}xst$ and u if α separates t and u and $x \notin \mathcal{V}t + \mathcal{V}u$.
3. $\alpha^x_\top$ separates $\mathsf{C}xst$ and $\mathsf{C}xuv$ if α separates s and u and $x \notin \mathcal{V}s + \mathcal{V}u$.
4. $\alpha^x_\bot$ separates $\mathsf{C}xst$ and $\mathsf{C}xuv$ if α separates t and v and $x \notin \mathcal{V}t + \mathcal{V}v$.

We now know that every expression is equivalent to a unique prime tree and that η computes this prime tree. We speak of the **prime tree for an expression**.

Corollary 26 prime trees are equivalent if and only if they are identical.

Theorem 27 (Decidability)

1. $\eta s = \eta t$ is decidable.

2. $s \approx t$ if and only if $\eta s = \eta t$.
3. s and t are separable if and only if $\eta s \neq \eta t$.
4. Equivalence and separability are decidable.

Proof Claim (1) holds since equality of conditional expressions is decidable. Claim (4) follows from claims (1), (2), and (3).

Let $\eta s = \eta t$. Then nontrivial directions of (2) and (3) follow with the correctness theorem.

Let $\eta s \neq \eta t$. Then nontrivial directions of (2) and (3) follow with the separation and the correctness theorem. ∎

Corollary 28 Two expressions are separable if and only if they are not semantically equivalent.

Proof Follows with (2) and (3) of the decidability theorem. ∎

Corollary 29 If s is prime, then $\eta s = s$.

Proof Let s be prime. Proof by contradiction (equality of expressions is decidable). Let $\eta s \neq s$. We have $\eta s \approx s$ and ηs is prime by the correctness theorem. By the separation theorem ηs and s are separable. Contradiction. ∎

Corollary 30 (Idempotence) $\eta(\eta s) = \eta s$

Corollary 31 If $s \approx t$ and t is prime, then $\eta s = t$.

Corollary 32 s is valid if and only if $\eta s = \top$.

Proof Follows with the decidability theorem and (4) of Fact 4 ∎

Corollary 33 s is unsatisfiable if and only if $\eta s = \bot$.

Proof Follows with the decidability theorem and (5) of Fact 4 ∎

We conclude with a few informal remarks. The prime tree for an expression represents the semantic information contained in the expression and omits all syntactic information. The prime representation of the semantic information is unique in that two expressions are semantically equivalent if and only if they have the same prime tree. We can see the prime tree for s as the semantic object **denoted** by s. Under this view semantic equivalence is **denotational** in that two expressions are semantically equivalent if and only if they denote the same prime tree.

We have mentioned boolean functions as an alternative denotational semantics for conditional expressions. The advantage of the prime tree semantics over the functional semantics is that prime trees are data objects with decidable equality while functions are abstract objects with undecidable equality.

Fact 34 There is a function that given a prime tree yields a satisfying assignment if the prime tree is different from $\bot$.

Fact 35 (Separation Function) There is a function that given two expressions constructs a separating assignment if the expressions are different prime trees.

Exercise 36 Design and verify functions as specified by Facts 34 and 35.

Exercise 37 (Strictly Sorted Lists) Recall that two lists are equivalent if they contain the same elements. There is an interesting parallel between prime trees and strictly sorted lists of numbers: Every list of numbers is equivalent to exactly one strictly sorted list. A strict sorting function is thus a function that computes for every list the unique strictly sorted normal form.

a) Define a predicate strictly sorted for lists of numbers.

b) Define a function *sort* that yields for every list of numbers a strictly sorted list that is equivalent.

c) Prove that different strictly sorted lists are not equivalent.

6 Significant Variables

A variable x is **significant** for an expression s if $s^x_\top$ and $s^x_\bot$ are separable. We will show that a variable is significant for an expression s if and only if it occurs in the prime tree of the expression.

Fact 38 Every significant variable of an expression occurs in the expression.

Fact 39 Significant variables are stable under semantic equivalence.

Fact 40 Every variable that occurs in a prime tree is significant for the expression.

Proof Let s be a prime tree and $x \in \mathcal{V}s$. We construct by induction on the primeness of s an assignment α separating $s^x_\top$ and $s^x_\bot$. We have $s = \mathsf{C}yuv$ since $x \in \mathcal{V}s$. Case analysis.

$x = y$. Since u and v are different prime trees, we have a separating assignment α for u and v by the separation theorem. Since x occurs neither in u nor in v, we have $\hat{\alpha}(u^x_\top) \neq \hat{\alpha}(v^x_\top)$. The claim follows since α separates $s^x_\top$ and $s^x_\bot$.

$x \neq y$. We assume $x \in \mathcal{V}u$ without loss of generality. By the inductive hypothesis we have a separating assignment α for $u^x_\top$ and $u^x_\bot$. The claim follows since $\alpha^y_\top$ separates $s^x_\top$ and $s^x_\bot$. ∎

$$\frac{}{\mathsf{C}\top st \equiv s} \qquad \frac{}{\mathsf{C}\bot st \equiv t} \qquad \frac{}{s \equiv \mathsf{C}x(s^x_\top)(s^x_\bot)}$$

$$\frac{s \equiv t}{t \equiv s} \qquad \frac{s \equiv t \qquad t \equiv u}{s \equiv u} \qquad \frac{s \equiv s' \qquad t \equiv t' \qquad u \equiv u'}{\mathsf{C}stu \equiv \mathsf{C}s't'u'}$$

Figure 1: Proof system for axiomatic equivalence of conditional expressions

Fact 41 The significant variables of an expression are exactly the variables that appear in the prime tree of the expression.

Exercise 42 Write a function that for an expression yields a list containing exactly the significant variables of the expression. Prove the correctness of the function.

7 Proof System for Conditional Expressions

Semantic equivalence of conditional expressions can be characterized with an equational proof system. We consider the proof system shown in Figure 1. In type theory, the proof system is formalized as an inductive definition of a predicate $s \equiv t$. We say that s and t are **axiomatically equivalent** if $s \equiv t$ is provable. The proof system consists of two evaluation rules for conditionals, a Shannon expansion rule, and three standard rules for equational deduction (symmetry, transitivity, congruence). There is no rule for reflexivity since reflexivity is derivable.

Fact 43 (Reflexivity) $s \equiv s$.

Proof The evaluation rule for $\top$ yields $\mathsf{C}\top st \equiv s$. The claim $s \equiv s$ folows with symmetry and transitivity. ∎

Fact 44 (Soundness) If $s \equiv t$, then $s \approx t$.

Proof By induction on the derivation of $s \equiv t$. ∎

Fact 45 (Consistency) $\top \not\equiv \bot$.

Completeness of axiomatic equivalence (the converse of soundness) follows from the fact that $\eta s \equiv s$. In fact, our proof system for axiomatic equivalence is designed so that we have $\eta s \equiv s$.

Fact 46 (Ground Evaluation) $s \equiv \hat{\alpha}\, s$ if s is ground.

Proof By induction on s using the evaluation rules for conditionals. ∎

Fact 47 $red\ x\ s\ t \equiv \mathsf{C}xst$ if $x \notin \mathcal{V}s$.

Proof The critical case is $s = t$. Then $red\ x\ s\ s = s \equiv \mathsf{C}x(s^x_\top)(s^x_\bot) = \mathsf{C}xss$ since $x \notin \mathcal{V}s$. ∎

Lemma 48 $\eta' As \equiv s$ if $\mathcal{V}s \subseteq A$.

Proof Follows by induction on s with Facts 46 and 47 and Theorem 23. ∎

Theorem 49 (Axiomatic Correctness) $\eta s \equiv s$.

Proof Follows with Lemma 48. ∎

Theorem 50 (Informative Completeness) There is a function that given two conditional expressions s and t constructs either a derivation of $s \equiv t$ or a separating assignment for s and t.

Proof Case analysis on $\eta s = \eta t$. If $\eta s = \eta t$, then $s \equiv t$ by the equivalence theorem. If $\eta s \neq \eta t$, then s and t are separable by the correctness of η and the separation function. ∎

Corollary 51 (Decidability) Axiomatic equivalence of conditional expressions is decidable.

Proof Follows with informative completeness and soundness. ∎

Corollary 52

1. $s \approx t$ if and only if $s \equiv t$.
2. $s \not\approx t$ if and only if $s \not\equiv t$.

8 Axiomatic Separation

The proof system for axiomatic equivalence has an important property we call axiomatic separability. Axiomatic separability says that negated propositions $s \not\equiv t$ have axiomatic proofs based on variable substitution.

We define a **substitution operator** $\tilde{\alpha}s$ that given an assignment α and an expression s yields the expression obtained from s by replacing every variable occuring in s according to the assignment α.

$$\tilde{\alpha}\top := \top \qquad \tilde{\alpha}\bot := \bot \qquad \tilde{\alpha}x := \alpha x \qquad \tilde{\alpha}(\mathsf{C}stu) := \mathsf{C}(\tilde{\alpha}s)(\tilde{\alpha}t)(\tilde{\alpha}u)$$

The substitution operator treats an assignment as a substitution that maps every variable to either the expression $\top$ or the expression $\bot$.

Fact 53 (Ground Evaluation) $\tilde{\alpha}s$ is ground and $\tilde{\alpha}s \equiv \hat{\alpha}s$.

Proof By induction on s using the evaluation rules. ∎

Fact 54 (Axiomatic Separation) An assignment α separates two conditional expressions s and t if and only if either $\tilde{\alpha}s \equiv \top$ and $\tilde{\alpha}t \equiv \bot$ or $\tilde{\alpha}s \equiv \bot$ and $\tilde{\alpha}t \equiv \top$.

9 Example: Diet Rules

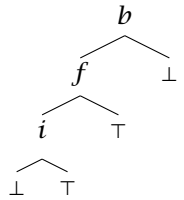
On a TV show a centenarian is asked for the secret of his long life. Oh, he says, my long life is due to a special diet that I started 60 years ago and follow by every day. The presenter gets all excited and asks for the diet. Oh, that's easy, says the old gentleman, there are only 3 rules you have to follow:

1. If you don't take beer, you must have fish.
2. If you have both beer and fish, don't have ice cream.
3. If you have ice cream or don't have beer, then don't have fish.

Let's look at the diet rules from a logical perspective. Obviously, the diet is only concerned with three boolean properties of a meal: having beer, having fish, and having ice cream. We can model these properties with three boolean variables b, f, i and describe the diet with a boolean expression that evaluates to true if and only if the diet is observed by a meal:

$$(\neg b \to f) \ \wedge \ (b \wedge f \to \neg i) \ \wedge \ (i \vee \neg b \to \neg f)$$

The expression is one possible description of the diet. The prime tree for the expression is a more explicit description of the diet:



It tells us that the diet is observed if and only if the following rules are observed:

1. Always drink beer.
2. Do not have both fish and ice cream.

Clearly, the prime tree represents the diet more explicitly than the rules given by the gentleman. From the prime tree we learn that we the diet can be described compactly with the expression $b \wedge \neg(f \wedge i)$.

Exercise 55 Four girls agree on some rules for a party:

$$
\begin{aligned}
x \wedge y &= y \wedge x & \qquad x \vee y &= y \vee x \\
x \wedge \top &= x & \qquad x \vee \bot &= x \\
x \wedge \neg x &= \bot & \qquad x \vee \neg x &= \top \\
x \wedge (y \vee z) &= (x \wedge y) \vee (x \wedge z) & \qquad x \vee (y \wedge z) &= (x \vee y) \wedge (x \vee z) \\
\top &\neq \bot
\end{aligned}
$$

Figure 2: Axioms for boolean algebras

i) Whoever dances which Richard must also dance with Peter and Michael.

ii) Whoever does not dance with Richard is not allowed to dance with Peter and must dance with Chris.

iii) Whoever does not dance with Peter is not allowed to dance with Chris.

Express these rules as simply as possible.

1. Describe each rule with a boolean formula. Use the variables c (Chris), p (Peter), m (Michael), and r (Richard).
2. Give the prime tree for the conjunction of the rules. Use the order $c < p < m < r$.

10 Boolean Algebra

Boolean algebra is a mathematical theory that originated with the work of George Boole [2] in 1847. It seems to be the first abstract algebraic theory ever considered. Another abstract algebraic theory is group theory.

Boolean algebra is centered around the notion of a boolean algebra. A **boolean algebra** consists of a set X called **carrier** and five **operations** $\top : X$, $\bot : X$, $\neg : X \to X$, $\wedge : X \to X \to X$, and $\vee : X \to X \to X$ satisfying the **axioms** shown in Figure 2. The constants $\top$ and $\bot$ are called operations as a matter of convenience.

A prominent example of a boolean algebra is the **two-valued algebra** $\mathcal{T}$ obtained with the two-valued type **B** and the operations defined in Section 2.

A prominent class of boolean algebras are the power set algebras. The **power set algebra** $\mathcal{P}_X$ for a nonempty set X takes the power set $\mathcal{P}X$ as carrier, $\emptyset$ as $\bot$, X as $\top$, set complement as $\neg$, set intersection as $\wedge$, and set union as $\vee$. It is straightforward to verify that the axioms for boolean algebras are satisfied by each power set algebra.

More generally, any subsystem of a power set algebra containing the empty set and being closed under complement, intersection, and union yields a boolean algebra.

There are different axiomatizations of boolean algebras in the literature [6]. Our axiomatization is a reformulation of the axiomatization used in Whitesitt's [7] textbook, which in turn is a variant of an axiomatization devised by Huntington [5] in 1904. Huntington [5] discovered that associativity, which until then appeared as an axiom, can be derived from the other axioms. Boolean algebra originated with Boole [2] and seems to be the first abstract algebra ever considered. Boole thought of conjunction as class intersection and of disjunction as union of disjoint classes [1].

Our main interest in Boolean algebra is to understand which equations one can prove or disprove in a boolean algebra. The starting point are the axioms for boolean algebras. Every equation that can be obtained with the basic equational proof rules from the axioms for boolean algebra must hold for every boolean algebra. For instance, one can show this way that conjunction and disjunction are associative in every boolean algebra. A good reason for having boolean algebra in the mathematical curriculum is that is provides a straightforward playground for presenting equational deduction and abstract algebraic methods.

It turns out that the boolean axioms are complete for boolean algebras. That is, given any boolean algebra, an equation formed with the operations of the algebra holds in the algebra if and only if it can be derived from the axioms for boolean algebras with the basic equational proof rules. This is a remarkable result given that there are rather different boolean algebras. It tells us that an equation holds in a power set algebra if and only if it holds in the two-valued algebra. The reduction to the two-valued algebra gives us a decision algorithm for equations over arbitrary power set algebras. It also gives us a decision method for whether or not an equation can be derived from the axioms for boolean algebras with the basic equational proof rules.

Whitesitt [7] gives an elementary mathematical development of Boolean algebra. As is typical for the mathematical literature on boolean algebra, his development is rather informal since at no point he introduces expressions as formal mathematical objects. If we want a formal (i.e., computer-checkable) account of Whitesitt's development, we are forced to introduce variables, expressions, assignments, and evaluation functions as formal objects.

The formal treatment of syntax is standard in computer science. Formal syntax is a prerequisite for the investigation of proof systems. It is also the prerequisite for the theory of computation that originated with Gödel and Turing.

11 Proof System for Boolean Algebra

We now start a formal investigation of boolean algebra. We will focus on the equational proof system for boolean algebra. Most results for boolean algebra

$$s \wedge t \equiv t \wedge s \qquad s \vee t \equiv t \vee s$$

$$s \wedge \top \equiv s \qquad s \vee \bot \equiv s$$

$$s \wedge \neg s \equiv \bot \qquad s \vee \neg s \equiv \top$$

$$s \wedge (t \vee u) \equiv (s \wedge t) \vee (s \wedge u) \qquad s \vee (t \wedge u) \equiv (s \vee t) \wedge (s \vee u)$$

$$\frac{s \equiv t}{t \equiv s} \qquad \frac{s \equiv t \quad t \equiv u}{s \equiv u} \qquad \frac{s \equiv s'}{\neg s \equiv \neg s'} \qquad \frac{s \equiv s' \quad t \equiv t'}{s \wedge t \equiv s' \wedge t'} \qquad \frac{s \equiv s' \quad t \equiv t'}{s \vee t \equiv s' \vee t'}$$

Figure 3: Proof system for axiomatic equivalence of boolean expressions

are either results for the proof system or straightforward consequences of results for the proof system. Our main result will establish the correctness of the prime tree method for boolean algebra.

We now consider **boolean expressions**

$$s, t, u, v \ ::= \ \top \mid \bot \mid x \mid \neg s \mid s \wedge t \mid s \vee t \qquad (x \in \mathbf{N})$$

obtained with $\top$, $\bot$, variables, negation, conjunction, and disjunction.

We define **axiomatic equivalence** $s \equiv t$ of boolean expressions with the equational proof system shown in Figure 3. The lines of the premise-free rules are omitted. The premise-free rules assert the axioms for boolean algebras (**commutativity**, **identity**, **negation**, and **distributivity**). The rules with premises provide the standard machinery of equational deduction (**symmetry**, **transitivity**, and **congruence**). We will refer to the premise-free rules as **axioms** and to the rules with premises as **proper rules**.

A standard rule for equational deduction is reflexivity. We have omitted reflexivity from the proof system since it can be derived with the other rules.

Fact 56 (Reflexivity) $s \equiv s$.

Proof Follows with identity, symmetry, and transitivity. ∎

The congruence rules of the proof system make it possible to rewrite with equivalences. In Coq, one registers axiomatic equivalence as an equivalence relation with setoid rewriting. This takes care of reflexivity, symmetry, and transitivity and provides for rewriting with equivalences at the top level. One also registers the congruence rules so that one can rewrite with equivalences below negations, conjunctions, and disjunctions.

There is a symmetry present in the boolean axioms known as *duality*. The symmetry holds for all derivable equivalences. We make the symmetry precise

with a **dualizing function** $\hat{s}$ that maps a boolean expression to a boolean expression called its **dual**.

$$\hat{\top} := \bot \qquad \hat{x} := x \qquad \widehat{s \wedge t} := \hat{s} \vee \hat{t}$$
$$\hat{\bot} := \top \qquad \widehat{\neg s} := \neg \hat{s} \qquad \widehat{s \vee t} := \hat{s} \wedge \hat{t}$$

The dualizing function $\hat{s}$ is self-inverting.

Fact 57 (Involution) $\hat{\hat{s}} = s$.

Theorem 58 (Duality) $s \equiv t$ if and only if $\hat{s} \equiv \hat{t}$.

Proof Let $s \equiv t$. We show $\hat{s} \equiv \hat{t}$ by induction on the derivation of $s \equiv t$. The proof is straightforward. The direction from right to left follows with involution and the direction already shown. ∎

The duality theorem tells us that the rule

$$\frac{\hat{s} \equiv \hat{t}}{s \equiv t} \quad \textbf{duality}$$

is **admissible**. Admissibility of a rule means that there is a function that constructs a derivation of the conclusion given derivations for the premises. Such a function is constructed in the proof of the duality theorem. The duality rule will be extremely useful in the following.

Constructing proofs in a formal proof system becomes much easier once the right admissible rules are established. In general, one wants to define a proof system with as few rules as possible since every defining rule needs to be accounted for in every admissibility proof. Given that when we construct proofs we can use admissible rules as if they were defining rules, working with few defining rules does not result in a loss of convenience.

Fact 59 (Evaluation Laws)

$\neg\top \equiv \bot$	$\neg\bot \equiv \top$	**constant negation**
$s \wedge \top \equiv s$	$s \vee \bot \equiv s$	**identity**
$s \wedge \bot \equiv \bot$	$s \vee \top \equiv \top$	**annulation**

Proof By duality it suffices to show the equivalences on the left. The identity law is an axiom. Here is the proof of constant negation.

$\neg\top \equiv \neg\top \wedge \top$	identity
$\equiv \top \wedge \neg\top$	commutativity
$\equiv \bot$	negation

Note that the rules for symmetry and transitivity are used tacitly. In Coq one can do the above proof with setoid rewriting. The proof for annulation starts with $\bot \equiv s \wedge \neg s \equiv s \wedge (\neg s \vee \bot)$. ∎

Theorem 60 (Ground Evaluation) If s is ground, then $s \equiv \top$ or $s \equiv \bot$.

Proof By induction on s using the evaluation laws. ∎

Fact 61

$s \wedge s \equiv s$	$s \vee s \equiv s$	**idempotence**
$s \wedge (s \vee t) \equiv s$	$s \vee (s \wedge t) \equiv s$	**absorption**

Proof By duality it suffices to show the equivalences on the left. The proofs are straightforward if one knows the trick. Here is the proof of idempotence.

$s \equiv s \wedge \top$	identity
$\equiv s \wedge (s \vee \neg s)$	negation
$\equiv s \wedge s \vee s \wedge \neg s$	distribution
$\equiv s \wedge s \vee \bot$	negation
$\equiv s \wedge s$	identity

Note that the rules for symmetry, congruence, reflexivity, and transitivity are used tacitly. In Coq one can do the above proof with setoid rewriting. ∎

Fact 62 (Expansion) $s \equiv (t \vee s) \wedge (\neg t \vee s)$ and $s \equiv (t \wedge s) \vee (\neg t \wedge s)$.

Fact 63 (Expansion) The following rule is admissible.

$$\frac{u \vee s \equiv u \vee t \qquad \neg u \vee s \equiv \neg u \vee t}{s \equiv t}$$

Fact 64 (Associativity) $s \wedge (t \wedge u) \equiv (s \wedge t) \wedge u$ and $s \vee (t \vee u) \equiv (s \vee t) \vee u$.

Proof By duality it suffices to show the left equivalence. By expansion it suffices to show the following equivalences:

$$s \vee s \wedge (t \wedge u) \equiv s \vee (s \wedge t) \wedge u$$
$$\neg s \vee s \wedge (t \wedge u) \equiv \neg s \vee (s \wedge t) \wedge u$$

The first equivalence follows with absorption and distributivity (both sides reduce to s). The second equivalence follows with distributivity, negation, and idenditiy (both sides reduce to $(\neg s \vee t) \wedge (\neg s \vee u)$). ∎

Fact 65 (Uniqueness of Complements) The following rule is admissible.

$$\frac{s \wedge t \equiv \bot \qquad s \vee t \equiv \top}{\neg s \equiv t}$$

Proof Let $s \wedge t \equiv \bot$ and $s \vee t \equiv \top$. We have $\neg s \equiv \neg s \wedge (s \vee t) \equiv \neg s \wedge s \vee \neg s \wedge t \equiv \neg s \wedge t$ and $t \equiv t \wedge (s \vee \neg s) \equiv t \wedge s \vee t \wedge \neg s \equiv t \wedge \neg s$. ∎

Fact 66 (Double Negation) $\neg\neg s \equiv s$.

Proof Follows with uniqueness of complements and the negation rules. ∎

Fact 67 (De Morgan) $\neg(s \wedge t) \equiv \neg s \vee \neg t$ and $\neg(s \vee t) \equiv \neg s \wedge \neg t$.

Proof The first equivalence follows with uniqueness of complements and associativity. The second equivalence follows with duality. ∎

Exercise 68 Our axioms for boolean algebras (see Figure 2) are not independent. In November 2014 Fabian Kunze discovered that either of the two identity axioms can be derived from the other axioms.

a) Prove the annulation law $s \vee \top \equiv \top$ with the commutativity and identity axiom for conjunctions and the negation and distributivity axiom for disjunctions.

b) Prove the identity law $s \vee \bot \equiv s$ with the annulation law for disjunctions shown in (a), the identity, negation, and distributivity axiom for conjunctions, and the commutativity axiom for disjunctions.

12 Shannon Expansion for Boolean Algebra

We now show that Shannon's expansion law can be shown with the axioms of boolean algebra.

Lemma 69 (Propagation)

1. $u \wedge \neg s \equiv u \wedge \neg(u \wedge s)$
2. $u \wedge (s \wedge t) \equiv (u \wedge s) \wedge (u \wedge t)$

Proof Follows with de Morgan and associativity. ∎

We define the **substitution operation** s^x_t for boolean expressions as one would expect from the definition for conditional expressions.

$$\top^y_u := \top \qquad x^y_u := \text{if } x{=}y \text{ then } u \text{ else } x \qquad (s \wedge t)^y_u := s^y_u \wedge t^y_u$$
$$\bot^y_u := \bot \qquad (\neg s)^y_u := \neg(s^y_u) \qquad (s \vee t)^y_u := s^y_u \vee t^y_u$$

Theorem 70 (Shannon)

1. $x \wedge s \equiv x \wedge s^x_\top$
2. $\neg x \wedge s \equiv \neg x \wedge s^x_\bot$
3. $s \equiv x \wedge s^x_\top \vee \neg x \wedge s^x_\bot$

Proof

1. Follows by induction on s. The base cases are straightforward. The inductive cases follow by rewriting with the propagation equivalences (twice, forth and back) and the inductive hypotheses. Here is the rewrite chain for negation: $x \wedge \neg s \equiv x \wedge \neg(x \wedge s) \equiv x \wedge \neg(x \wedge s^x_\top) \equiv x \wedge \neg(s^x_\top) \equiv x \wedge (\neg s)^x_\top$.
2. Similar to (1).
3. Follows with (1) and (2): $s \equiv s \wedge (x \vee \neg x) \equiv x \wedge s \vee \neg x \wedge s \equiv x \wedge s^x_\top \vee \neg x \wedge s^x_\bot$. ∎

Exercise 71 (Replacement Rule) Consider the following rule:

$$\frac{s \equiv t}{u^x_s \equiv u^x_t} \quad \textbf{replacement}$$

1. Prove that the replacement rule is admissible.
2. Show that the congruence rules can be derived with the replacement rule.

13 Substitutivity for Boolean Algebra

There is a standard rule for equational deduction called substitutivity we have not mentioned so far. Substitutivity says that from an equivalence $s \equiv t$ one can derive every instance of $s \equiv t$, where an instance is obtained by instantiating the variables with terms. We now show that the rules of our proof system for boolean algebra can simulate the substitutivity rule.

A **substitution** is a function θ from variables to expressions. We define a **substitution operation** $\tilde{\theta}s$ that, given a substitution, maps boolean expressions to boolean expressions.

$$\begin{array}{lll} \tilde{\theta}\top := \top & \tilde{\theta}x := \theta x & \tilde{\theta}(s \wedge t) := \tilde{\theta}s \wedge \tilde{\theta}t \\ \tilde{\theta}\bot := \bot & \tilde{\theta}(\neg s) := \neg(\tilde{\theta}s) & \tilde{\theta}(s \vee t) := \tilde{\theta}s \vee \tilde{\theta}t \end{array}$$

Fact 72 (Substitutivity) If $s \equiv t$, then $\tilde{\theta}s \equiv \tilde{\theta}t$.

Proof By induction on the derivation of $s \equiv t$. Follows from the fact that the axioms of the proof system are closed under substitution. ∎

We now know that the substitutivity rule is admissible.

$$\frac{s \equiv t}{\widetilde{\theta} s \equiv \widetilde{\theta} t} \quad \textbf{substitutivity}$$

14 Consistency and Separation for Boolean Algebra

Recall that an assignment is a function from variables to booleans. We define an **evaluation function** $\hat{\alpha} s$ for assignments and boolean expressions as one would expect from the definition of the evaluation function for conditional expressions.

$$\begin{array}{lll} \hat{\alpha}\top := \top & \hat{\alpha} x := \alpha x & \hat{\alpha}(s \wedge t) := \hat{\alpha} s \wedge \hat{\alpha} t \\ \hat{\alpha}\bot := \bot & \hat{\alpha}(\neg s) := \neg(\hat{\alpha} s) & \hat{\alpha}(s \vee t) := \hat{\alpha} s \vee \hat{\alpha} t \end{array}$$

Fact 73 (Soundness) If $s \equiv t$, then $\hat{\alpha} s = \hat{\alpha} t$.

Proof By induction on the derivation of $s \equiv t$. ∎

Fact 74 (Consistency) $\top \not\equiv \bot$.

An assignment α separates two boolean expressions s and t if $\hat{\alpha} s \neq \hat{\alpha} t$. Two boolean expressions are **separable** if they are separated by some assignment.

Fact 75 (Disjointness)
Axiomatically equivalent boolean expressions are not separable.

We identify an assignment α with the substitution that maps a variable x to the expression $\top$ or $\bot$ if α maps x to the boolean $\top$ or $\bot$, respectively. In Coq, we realize the identification with a function that converts assignments into an substitutions.

Fact 76 (Ground Evaluation) $s \equiv \hat{\alpha} s$ if s is ground.

Proof Follows by induction on s using the evaluation laws (Fact 59). ∎

Fact 77 (Axiomatic Separation) An assignment α separates two boolean expressions s and t if and only if either $\tilde{\alpha} s \equiv \top$ and $\tilde{\alpha} t \equiv \bot$ or $\tilde{\alpha} s \equiv \bot$ and $\tilde{\alpha} t \equiv \top$.

Proof Follows with ground evaluation and consistency. ∎

$$
\begin{aligned}
\eta s &:= \eta'\,(sort\,(\mathcal{V}s))\,s \\
\eta'\;nil\;s &:= \widehat{\alpha_\bot}\,s \\
\eta'\;(x :: A)\;s &:= red\;x\;(\eta' A\,(s^x_\top))\;(\eta' A\,(s^x_\bot)) \\
red\;x\;s\;t &:= \mathsf{if}\;s = t\;\mathsf{then}\;s\;\mathsf{else}\;\mathsf{C}xst
\end{aligned}
$$

Figure 4: Prime tree procedure for boolean expressions

15 Prime Tree Procedure for Boolean Algebra

We now show that the prime tree method yields an informative decision procedure for axiomatic equivalence that in the positive case yields a derivation of the equivalence and in the negative case yields a separating assignment. There are only obvious modifications needed to adapt the prime tree procedure η from conditional expressions to boolean expressions. Figure 4 shows the prime tree procedure for boolean expressions.

For the correctness proof of the prime tree procedure, we define a **translation function** γs from conditional expressions to boolean expressions.

$$\gamma\top := \top \qquad \gamma\bot := \bot \qquad \gamma x := x \qquad \gamma(\mathsf{C}stu) := \gamma s \wedge \gamma t \vee \neg\gamma s \wedge \gamma u$$

Fact 78 (Preservation) $\hat{\alpha}(\gamma s) = \hat{\alpha} s$.

Theorem 79 (Correctness) ηs is prime, $\gamma(\eta s) \equiv s$, and $\mathcal{V}(\eta s) \subseteq \mathcal{V}s$.

Proof The proof is very similar to the correctness proofs for conditional expressions (Theorems 23 and 49). The adaptions concern the new lemmas for Shannon expansion (Theorem 70) and ground evaluation (Fact 76). ∎

Corollary 80 (Preservation) $\hat{\alpha}(\eta s) = \hat{\alpha} s$.

Proof Follows with soundness, Fact 78 and the correctness theorem. ∎

Theorem 81 (Informative Completeness) There is a function that given two boolean expressions s and t constructs either a derivation of $s \equiv t$ or a separating assignment for s and t.

Proof By case analysis on $\eta s = \eta t$. By the correctness theorem we know that ηs and ηt are prime and that $\gamma(\eta s) \equiv s$ and $\gamma(\eta t) \equiv t$.

Let $\eta s = \eta t$. Then $s \equiv t$.

Let $\eta s \neq \eta t$. The separation function gives us a separating assignment α for ηs and ηt. With preservation it follows that α separates s and t. ∎

Corollary 82 (Agreement) $s \equiv t$ if and only if $\hat{\alpha}s = \hat{\alpha}t$ for every assignment α.

Corollary 83 (Decidability) $s \equiv t$ is decidable.

Corollary 84 Two expressions are separable if and only if they are not axiomatically equivalent.

16 Soundness and Completeness for Boolean Algebras

We now relate the proof system for boolean algebra with abstract boolean algebras. For that we assume a carrier type X, operations $\top$, $\bot$, $\neg$, $\wedge$, and $\vee$, and proofs of the axioms for boolean algebras (see Figure 2). In Coq, we use a section to state the assumptions.

Note that we now have three readings of the symbols $\bot$ and $\top$: as booleans, as expressions, and as abstract values. Formally, of course, we need to have different names for different objects.

An **abstract assignment** is a function β mapping variables to values of X. We define an **evaluation operation** $\hat{\beta}s$ that, given an abstract assignment, maps every expression to a value of X.

$$\begin{array}{lll} \hat{\beta}\top := \top & \hat{\beta}x := \beta x & \hat{\beta}(s \wedge t) := \hat{\beta}s \wedge \hat{\beta}t \\ \hat{\beta}\bot := \bot & \hat{\beta}(\neg s) := \neg(\hat{\beta}s) & \hat{\beta}(s \vee t) := \hat{\beta}s \vee \hat{\beta}t \end{array}$$

Theorem 85 (Positive Soundness) If $s \equiv t$, then $\hat{\beta}s = \hat{\beta}t$.

Proof By induction on the derivation of $s \equiv t$. The proof is straightforward since the axioms of the proof system mirror the axioms we have assumed for the underlying algebra. ∎

The soundness theorem says that axiomatic equivalence implies equality in the abstract algebra. This means that every equivalence we have shown with the proof system carries over to the abstract algebra. This is essential for the proof of the following lemma.

We use $\dot{\alpha}$ to denote the abstract assignment obtained from an assignment α by replacing the booleans $\bot$ and $\top$ with the abstract values $\bot$ and $\top$.

Lemma 86 $\hat{\dot{\alpha}}s = \hat{\alpha}s$.

Proof By induction on s using soundness and the evaluation laws (Fact 59). ∎

Theorem 87 (Negative Soundness) If $\hat{\alpha}s \neq \hat{\alpha}t$, then $\hat{\dot{\alpha}}s \neq \hat{\dot{\alpha}}t$.

Proof Follows with Lemma 86. ∎

Theorem 88 (Agreement)
$s \equiv t$ if and only if $\hat{\beta}s = \hat{\beta}t$ for every abstract assignment β.

Proof One direction is positive soundness. The other direction follows with informative completeness and negative soundness. ∎

Exercise 89 Prove that two boolean expressions s and t are separable if and only if there exists an abstract assignment β such that $\hat{\beta}s \neq \hat{\beta}t$.

Exercise 90 (Independence of consistency axiom) Give an algebra satisfying all axioms for boolean algebras but $\top \neq \bot$.

Exercise 91 (Independence of negation axioms) Show that the negation axiom for conjunctions $s \wedge \neg s \equiv \bot$ cannot be derived from the other axioms. To do so, construct an algebra that dissatisfies the negation axiom for conjunctions but satisfies all other axioms for boolean algebras. Hint: A two-valued algebra where only negation deviates from the standard definition suffices.

Exercise 92 (Independence of distributivity axioms) Show that the distributivity axiom for conjunctions $s \wedge (t \vee u) \equiv s \wedge t \vee s \wedge u$ cannot be derived from the other axioms. To do so, construct a two-valued algebra that dissatisfies the distributivity axiom for conjunctions but satisfies all other axioms for boolean algebras. Hint: A two-valued algebra where only conjunction deviates from the standard definition suffices.

17 Summary: Equational Deduction for Boolean Algebra

It is clear that we can have different equational deduction systems for boolean algebra. A deduction system must provide for replacement of equals, reflexivity, symmetry, transitivity, and substitutivity. This can be accomplished following general principles not specific to boolean algebra. The rules specific for boolean algebra must provide for ground evaluation (Theorem 60) and Shannon expansion (Theorem 70).

Ground evaluation ensures that the abstract operations agree with the concrete boolean operations on $\top$ and $\bot$. Thus every ground expression is equivalent to either $\top$ or $\bot$ in accordance with concrete boolean evaluation. With Shannon expansion and ground evaluation it follows that expressions that have the same prime tree are equivalent. Expressions that have different prime trees can be separated with an assignment. By ground evaluation it follows that the expressions can be separated axiomatically.

18 Operations on Prime Trees

We will define operations on expressions that behave semantically like negation and conjunction and in addition map prime trees to prime trees. The precise specification of the operations appears in the correctness theorems 94 and 96. We could define the operations with η, for instance $not\ s = \eta(\neg s)$, but this is not what we are interested in. Instead, we will define the operations such that they correspond to the efficient operations for ordered and reduced BDDs. The correctness theorems for the operations will confront us with interesting verification problems.

The idea for the negation operation can be expressed with an obvious identity for booleans.

Fact 93 $\neg \mathsf{C} a b c = \mathsf{C} a (\neg b)(\neg c)$

This suggests that we can negate a prime tree by just replacing $\top$ and $\bot$ with each other. We define the operation *not* by structural recursion for all expressions.

$$\begin{array}{rcll} not\ \top & := & \bot & \\ not\ \bot & := & \top & \\ not\ (\mathsf{C} s t u) & := & \mathsf{C} s (not\ t)(not\ u) & \\ not\ x & := & \neg x & \text{otherwise} \end{array}$$

Theorem 94 (Correctness)

1. $\hat{\alpha}(not\ s) = \neg(\hat{\alpha} s)$.
2. $not(not\ s) = s$ if s is prime.
3. $not\ s \neq not\ t$ if $s \neq t$ and s and t are prime.
4. $\mathcal{V}(not\ s) \subseteq \mathcal{V} s$.
5. $not\ s$ is prime if s is prime.

We now come to the conjunction operation. The interesting case appears when both sides are prime conditionals. In this case one compares the two head variables and proceeds according to one the following boolean identities.

Fact 95 $\mathsf{C} a b c \wedge \mathsf{C} a d e = \mathsf{C} a (b \wedge d)(c \wedge e)$ and $\mathsf{C} a b c \wedge d = \mathsf{C} a (b \wedge d)(c \wedge d)$.

We define the operation *and* by nested structural recursion as follows:

$$
\begin{array}{rcll}
\mathit{and}\ \top\ t & := & t & \\
\mathit{and}\ \bot\ t & := & \bot & \\
\mathit{and}\ (\mathsf{C}xst)\ \top & := & \mathsf{C}xst & \\
\mathit{and}\ (\mathsf{C}xst)\ \bot & := & \bot & \\
\mathit{and}\ (\mathsf{C}xst)\ (\mathsf{C}yuv) & := & \mathit{red}\ x\ (\mathit{and}\ s\ u)\ (\mathit{and}\ t\ v) & \text{if } x = y \\
\mathit{and}\ (\mathsf{C}xst)\ (\mathsf{C}yuv) & := & \mathit{red}\ x\ (\mathit{and}\ s\ (\mathsf{C}yuv))\ (\mathit{and}\ t\ (\mathsf{C}yuv)) & \text{if } x < y \\
\mathit{and}\ (\mathsf{C}xst)\ (\mathsf{C}yuv) & := & \mathit{red}\ y\ (\mathit{and}\ (\mathsf{C}xst)\ u)\ (\mathit{and}\ (\mathsf{C}xst)\ v) & \text{if } x > y \\
\mathit{and}\ s\ t & := & s \wedge t & \text{otherwise}
\end{array}
$$

Theorem 96 (Correctness)

1. $\hat{\alpha}(\mathit{and}\ s\ t) = \hat{\alpha}s \wedge \hat{\alpha}t$.
2. $\mathcal{V}(\mathit{and}\ s\ t) \subseteq \mathcal{V}s \cup \mathcal{V}t$.
3. $\mathit{and}\ s\ t$ is prime if s and t are prime.

Given *not* and *and*, we can write another normalizer for expressions.

$$
\begin{array}{rcl}
\mu\top & := & \top \\
\mu\bot & := & \bot \\
\mu x & := & \mathsf{C}x\top\bot \\
\mu(\mathsf{C}stu) & := & \mathit{or}\ (\mathit{and}\ (\mu s)\ (\mu t))\ (\mathit{and}\ (\mathit{not}\ (\mu s))\ (\mu u)) \\
\mathit{or}\ s\ t & := & \mathit{not}\ (\mathit{and}\ (\mathit{not}\ s)\ (\mathit{not}\ t))
\end{array}
$$

Theorem 97 (Correctness) μs is prime and semantically equivalent to s.

19 Further Reading

Boolean algebra originated with the work of George Boole [2]. Whitesitt [7] gives an elementary mathematical introduction to boolean algebra. As is common in mathematics, Whitesitt treats expressions completely informal. That is, expressions do not appear as mathematical objects. Barnett [1] reviews the history of boolean algebra and its application in circuit design.

There are different axiomatizations of boolean algebras in the literature [5, 6]. We use a variant of Huntington's [5] axiomatization from 1904, which is also used in Whitesitt [7]. Huntington [5] shows that the axioms of his axiomatizations are independent. This is not the case for our axiomatization since either of the identity axioms is derivable from the other axioms (see Exercise 68). There

are straightforward independence proofs for the remaining axioms but commutativity. It seems that Huntington's axioms are cleverly formulated such that the independence proofs are easy. Huntington's axiomatization requires the existence of neutral elements and negations but does not require fixed constants for neutral elements and negation.

Ordered and reduced binary decision diagrams were invented by Bryant [3] as an efficient data structure for boolean functions as used in digital system design. Our definition of prime trees and the concomitant operations are derived from Bryant's work [3].

References

[1] Janet Heine Barnett. Applications of boolean algebra: Claude Shannon and circuit design. Internet, 2011.

[2] George Boole. *An Investigation of the Laws of Thought*. Reprinted by Merchant Books, 2010. Walton, London, 1847.

[3] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[4] Robert L. Constable. Computational type theory. *www.scholarpedia.org*, 2009.

[5] Edward V. Huntington. Sets of independent postulates for the algebra of logic. *Transactions of the American Mathematical Society*, 5(3):288–309, 1904.

[6] Edward V. Huntington. New sets of independent postulates for the algebra of logic, with special reference to Whitehead and Russell's Principia Mathematica. *Trans. Amer. Math. Soc.*, 35:274–304, 1933.

[7] John Eldon Whitesitt. *Boolean Algebra and Its Applications*. Reprinted by Dover, 2010. Addison-Wesley, 1961.