Boolean Logic

Lecture Notes

Gert Smolka
Saarland University
November 22, 2016

Abstract. We study satisfiability and equivalence of boolean expressions. We show that prime trees (reduced and ordered decision trees) are a unique normal form for boolean expressions. We verify a SAT solver and a prime tree normaliser. We show that assignment equivalence agrees with the equivalence obtained from the axioms of boolean algebra.

The material is presented using type-theoretic language. There is an accompanying Coq development complementing the mathematical presentation given in the notes.

The notes also serve as introduction to formal syntax. Expressions are modelled with an inductive type and recursive functions and inductive predicates on expressions are studied. This includes notions like substitution and equational deduction.

Advice to the student. To master the material, you want to understand both the notes and the accompanying Coq development. It is important that you can switch between the informal mathematical presentation of the notes and the formal development in Coq. You may think of the notes as the design of the theory and the Coq development as the implementation of the theory. The goal is to understand both the design and the implementation of the theory. The notes occasionally omit details that are spelled out in the Coq development (e.g., the precise definition of substitution). Make sure you can fill in the missing details.

Contents

1 Boolean Operations

We start with an inductive type $\mathbb{B} := \top \mid \bot$ with two elements $\top$ and $\bot$ (read bool, true and false). The native operation on $\mathbb{B}$ is the boolean match

$$\mathsf{match}\ s_0\ \mathsf{with}\ \top \Longrightarrow s_1 \mid \bot \Longrightarrow s_2$$

which may be written with the if-then-else notation:

$$\mathsf{if}\ s_0\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2$$

We assume that a, b, c are variables ranging over $\mathbb{B}$ and define the following boolean operations:

$$\begin{array}{rcll} Cabc & := & \mathsf{if}\ a\ \mathsf{then}\ b\ \mathsf{else}\ c & \textit{conditional} \\ a \wedge b & := & Cab\bot & \textit{conjunction} \\ a \vee b & := & Ca\top b & \textit{disjunction} \\ a \to b & := & Cab\top & \textit{implication} \\ \neg a & := & Ca\bot\top & \textit{negation} \end{array}$$

Observe that the boolean conditional $Cs_0s_1s_3$ is weaker than the boolean match since s_2 and s_3 must have type $\mathbb{B}$. Also note that the boolean conditional can express all other boolean operations (together with $\bot$ and $\top$).

Conjunction and negation can express disjunction:

$$a \vee b = \neg(\neg a \wedge \neg b)$$

Since we have

$$Cabc = a \wedge b \vee \neg a \wedge c$$

conjunction and negation can express the boolean conditional. Similarly, disjunction and negation can express the boolean conditional. Moreover, implication and false can express the boolean conditional.

Exercise 1 Carry out the above definitions in Coq and prove the claimed equations. Boolean case analysis and definitional equality suffice for the proofs.

Exercise 2

a) Show that conjunction and negation can express $\bot$ and $\top$.

b) Show that disjunction and negation can express the boolean conditional.

c) Show that implication $a \to b$ and $\bot$ can express the boolean conditional.

d) Show that the operation $a \barwedge b := \neg(a \wedge b)$ (*nand*) can express the boolean conditional and also $\bot$ and $\top$.

2 SAT Solver

The boolean satisfiability problem (SAT) asks whether the values of the variables of a given boolean expression can be chosen such that the formula evaluates to true. For instance, the expression $x \wedge \neg y$ evaluates to true with x = true and y = false. In contrast, the expression $x \wedge \neg x$ evaluates to false no matter how the value of x is chosen.

An assignment is a function that assigns a boolean (i.e., true or false) to every variable. An assignment satisfies a boolean expression if the expression evaluates to true under the assignment, and an expression is satisfiable if it has a satisfying assignment.

A SAT solver is a procedure that decides whether a given boolean expression is satisfiable and returns a satisfying assignment in the positive case. We will design and verify a simple SAT solver.

A SAT solver requires a representation of boolean expressions as data. We represent boolean expressions as the values of an inductive type Exp providing constructors for $\top$, $\bot$, variables, and the boolean conditional. Variables are represented as numbers.

An expression not containing variables is called ground. The evaluation of a ground expression does not depend on the assignment. We have

$$\text{ground } s \;\to\; \alpha s = \beta s \tag{1}$$

for all assignments α and β. We write αs for the boolean to which the expression s evaluates under the assignment α. Evaluation is defined by recursion on expressions. We also have

$$\text{ground } s \;\to\; \text{sat } s \;\leftrightarrow\; \alpha s = \top \tag{2}$$

A SAT solver for ground expressions is now straightforward. It evaluates a given expression s with some assignment, say $\tau = \lambda x.\top$. If $\tau s = \top$, it returns the satisfying assignment τ. Otherwise, the expression is unsatisfiable and the solver returns the negative answer.

For expressions containing variables we will employ a variable elimination technique based on the fact that an expression s is satisfiable if and only if one of the two instantiations $s^x_\top$ and $s^x_\bot$ is satisfiable:

$$\text{sat } s \;\leftrightarrow\; \text{sat } s^x_\top \;\vee\; \text{sat } s^x_\bot \tag{3}$$

The expressions $s^x_\top$ and $s^x_\bot$ are obtained from the expression s by replacing the variable x with the expressions $\top$ and $\bot$, respectively. One speaks of substitution.

Given a satisfying assignment for one of the instantiations $s^x_\top$ and $s^x_\bot$, we can get a satisfying assignment for s:

$$\alpha(s^x_\top) = \top \;\to\; \alpha^x_\top(s) = \top \tag{4}$$
$$\alpha(s^x_\bot) = \top \;\to\; \alpha^x_\bot(s) = \top \tag{5}$$

We write α^x_b for the assignment obtained from α by mapping x to the boolean b. The two implications are consequences of a more general fact we call **shift**:

$$\alpha(s^x_t) = \alpha^x_{\alpha t}(s) \tag{6}$$

Shift can be shown by induction on s.

We need a function $\mathcal{V}s$ that for an expression s yields a list containing exactly the variables occurring in s. The solver will recurse on the list $\mathcal{V}s$ and eliminate the variables occurring in s one after the other.

Given a type X, we write X^* for the type of lists over X and $X_\bot$ for the type of options over X. We realize the SAT solver with a function

$$\text{solve} : \text{Var}^* \to \text{Exp} \to \text{Assn}_\bot$$

satisfying the specification

$$\mathcal{V}s \subseteq A \;\to\; \mathsf{match}\ \text{solve}\ A\ s\ \mathsf{with}\ \lfloor \alpha \rfloor \Longrightarrow \alpha s = \top \mid \bot \Longrightarrow \neg\text{sat}\ s \tag{7}$$

Make sure that you understand the outlined design of the SAT solver. It is now routine to write the function solve:

$$\begin{aligned}
\text{solve}\ A\ s \;:=\; & \mathsf{match}\ A\ \mathsf{with} \\
& \mid \text{nil} \;\Longrightarrow\; \mathsf{if}\ \tau s\ \mathsf{then}\ \lfloor \tau \rfloor\ \mathsf{else}\ \bot \\
& \mid x :: A \;\Longrightarrow\; \mathsf{match}\ \text{solve}\ A\ (s^x_\top),\ \text{solve}\ A\ (s^x_\bot)\ \mathsf{with} \\
& \qquad \mid \lfloor \alpha \rfloor,\ _ \;\Longrightarrow\; \lfloor \alpha^x_\top \rfloor \\
& \qquad \mid _,\ \lfloor \alpha \rfloor \;\Longrightarrow\; \lfloor \alpha^x_\bot \rfloor \\
& \qquad \mid \bot,\ \bot \;\Longrightarrow\; \bot
\end{aligned}$$

With the given facts it is straightforward to verify that the function solve satisfies its specification (7). The proof is by induction on A. To apply the inductive hypothesis to the recursive calls of solve, we need an additional fact for substitutions

$$\text{ground}\ t \to \mathcal{V}s \subseteq x :: A \to \mathcal{V}(s^x_t) \subseteq A \tag{8}$$

which follows by induction on s (s and t are expressions).

Coincidence and Expansion

If you try to prove (3) in Coq, you will discover a difficulty with the direction from left to right. The problem is that Coq doesn't assume functional extensionality and so cannot prove $\alpha = \alpha^x_{\alpha x}$. The problem can be bypassed by making use of a fact known as **coincidence**:

$$(\forall x \in \mathcal{V}s.\ \alpha s = \beta s) \to \alpha s = \beta s \tag{9}$$

Coincidence says that the evaluation of an expression s under two assignments α and β agrees if α and β agree on all variables in s. Coincidence formalizes the fact that the evaluation of an expression depends only on the variables occurring in the expression. Note that coincidence generalizes Fact (1).

With shift and coincidence we can show a fact called **expansion**:

$$\alpha s = \mathsf{if}\ \alpha x\ \mathsf{then}\ \alpha(s^x_\top)\ \mathsf{else}\ \alpha(s^x_\bot) \tag{10}$$

We summarise the main results of this section with a couple of formal statements.

Fact 3 (Coincidence) $(\forall x \in \mathcal{V}s.\ \alpha s = \beta s) \to \alpha s = \beta s$.

Fact 4 (Shift) $\alpha(s^x_t) = \alpha^x_{\alpha t}(s)$.

Fact 5 (Expansion) $\alpha s = \mathsf{if}\ \alpha x\ \mathsf{then}\ \alpha(s^x_\top)\ \mathsf{else}\ \alpha(s^x_\bot)$.

Theorem 6 (SAT Solver) There is a function $\forall s.\ \{\alpha \mid \alpha s = \top\} + \{\neg \mathrm{sat}\ s\}$.

Corollary 7 (Decidability) Satisfiability of boolean expressions is decidable.

Exercise 8 Formalize the SAT solver in Coq. Proceed as follows.

a) Define the types for variables and assignments: $\mathrm{Var} := \mathbb{N}$ and $\mathrm{Assn} := \mathrm{Var} \to \mathbb{B}$.

b) Define an inductive type of Exp of boolean expressions. Make sure you have constructors for variables and booleans.

c) Define an evaluation function $\mathrm{eva} : \mathrm{Assn} \to \mathrm{Exp} \to \mathbb{B}$.

d) Define a substitution function $\mathrm{subst} : \mathrm{Exp} \to \mathrm{Var} \to \mathrm{Exp} \to \mathrm{Exp}$.

e) Define a function $\mathcal{V} : \mathrm{Exp} \to \mathrm{Var}^*$.

f) Define the assignment τ and a function $\mathrm{update} : \mathrm{Assn} \to \mathrm{Var} \to \mathbb{B} \to \mathrm{Var} \to \mathbb{B}$.

g) Define the function $\mathrm{solve} : \mathrm{Var}^* \to \mathrm{Exp} \to \mathrm{Assn}_\bot$.

h) Define the predicates sat and ground.

i) Prove the shift law: $\alpha(s^x_t) = \alpha^x_{\alpha t}(s)$.

j) Prove the coincidence law: $(\forall x \in \mathcal{V}s.\ \alpha s = \beta s) \to \alpha s = \beta s$.

k) Prove: $\alpha^x_{\alpha x}(z) = \alpha z$.

l) Prove the expansion law: $\alpha s = \mathsf{if}\ \alpha x\ \mathsf{then}\ \alpha(s^x_\top)\ \mathsf{else}\ \alpha(s^x_\bot)$.

m) Prove: ground $t \to \mathcal{V}s \subseteq x :: A \to \mathcal{V}(s^x_t) \subseteq A$.

n) Prove that solve is correct (i.e., Statement (7)).

o) Prove that satisfiability of expressions is decidable.

Exercise 9 Prove that $s^x_t = s$ if $x \notin \mathcal{V}s$.

Exercise 10 (Assignments as lists) A list A of variables represents an assignment

$$\alpha_A(x) := \mathsf{if}\ x \in A\ \mathsf{then}\ \top\ \mathsf{else}\ \bot$$

a) Give an assignment that cannot be represented as a list.

b) Show that an expression s is satisfiable if and only if there is a list $A \subseteq \mathcal{V}s$ such that the corresponding assignment α_A satisfies s.

c) There is the possibility of defining satisfiability based on list assignments. Construct a SAT solver not using substitution making use of this idea. Hint: The power list of $\mathcal{V}s$ provides a finite and complete candidate set for satisfying assignments.

Exercise 11 Equality of expressions is decidable. Make sure you know the proof. Prove the fact in Coq using the automation tactic *decide equality*.

3 Equivalence of Boolean Expressions

We assume that we have boolean expressions as follows:

$$s, t, u \ ::= \ \top \mid \bot \mid x \mid Cstu \qquad (x : \mathbb{N})$$

Further operation may be defined as abbreviations. We define negation as

$$\neg s \ := \ Cs\bot\top$$

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

$$s \approx t \ := \ \forall \alpha.\ \alpha s = \alpha t$$

Fact 12 Equivalence of expressions is an equivalence relation.

Fact 13 (Ground Evaluation) ground $s \to s \approx \mathsf{if}\ \alpha s\ \mathsf{then}\ \top\ \mathsf{else}\ \bot$.

Proof Case analysis on αs and coincidence. ∎

Fact 14 ground $s \to s \approx \top \vee s \approx \bot$.

Equivalence reduces to unsatisfiability.

Fact 15 $s \approx t \;\leftrightarrow\; \neg\mathrm{sat}(Cs(\neg t)t)$.

Fact 16 Equivalence of expressions is decidable.

An assignment α **separates** two expressions s and t if $\alpha s \neq \alpha t$.

Fact 17 An assignment α separates two expressions s and t if it satisfies $Cs(\neg t)t$. Hence separability is decidable.

Fact 18 Two expressions are separable iff they are not equivalent.

Proof The direction from right to left is interesting since we must show an existential claim from a negated universal quantification. This is possible constructively since separability is decidable and thus the proof can be done by contradiction. ∎

Exercise 19 Prove the following equivalences:

a) $\top \approx s \vee \neg s$ and $\bot \approx s \wedge \neg s$

b) $\neg\neg s \approx s$

c) $Cstt \approx t$ (reduction)

d) $\neg Cstu \approx Cs(\neg t)(\neg u)$

e) $Cst_1u_1 \wedge Cst_2u_2 \approx Cs(t_1 \wedge t_2)(u_1 \wedge u_2)$

f) $s \approx Cx(s^x_\top)(s^x_\bot)$ (expansion)

Exercise 20 Prove that satisfiability reduces to disequivalence: $\mathrm{sat}\, s \leftrightarrow s \not\approx \bot$.

Exercise 21 Give a function $f : \mathbb{N} \to \mathrm{Exp}$ such that fm and fn are separable whenever m and n are different.

Exercise 22 (Compatibility) Prove the following *compatibility properties*:

a) If $s \approx s'$, $t \approx t'$, and $u \approx u'$, then $Cstu \approx Cs't'u'$.

b) If $s \approx s'$, then $\neg s \approx \neg s'$.

c) If $s \approx s'$, then s is satisfiable iff s' is satisfiable.

Exercise 23 (Replacement) Prove that $s \approx t$ implies $s^x_u \approx t^x_u$.

Exercise 24 (Checking equivalence) Write a function that checks whether two expressions are equivalent. Prove the correctness of your function. Exploit a SAT solver and Fact 15.

Exercise 25 (Computing separating assignments) Write a function that for two expressions yields a separating assignment if there is one. Prove the correctness of your function. Exploit a SAT solver and Fact 17.

Exercise 26 (Validity) An expression is called *valid* if it evaluates to true with every assignment. Prove the following:

a) s valid iff $s \approx \top$.

b) s valid iff $\neg s$ is unsatisfiable.

c) s is satisfiable iff $\neg s$ is not valid.

d) Validity of boolean expressions is decidable.

4 Shallow and Deep Embedding

Many problems can be represented as boolean expressions such that the solutions of the problem appear as the solutions of the expression. Equivalence of boolean expressions is defined such that equivalent expressions represent the same problem. One often works with different but equivalent representations of a problem. One representation may describe the problem as it naturally appears while another representation may describe the problem such that the solution can be seen easily. The situation is familiar from equations over numbers, which are routinely used with technical and financial problems.

An important application of boolean logic is hardware design. There boolean expressions are used to describe boolean functions that are to be implemented with hardware. The idea is that a boolean expression describes a function that for the inputs identified by the variables occurring in the expression yields an output. For instance, $x \wedge y$ describes a boolean function that yields true if and only if both inputs x and y are true. The way a boolean function is described in a design may be rather different from the way it is implemented with hardware. It is the notion of equivalence that connects the description of a boolean function with its implementation.

We illustrate the use of boolean expressions and the notion of equivalence with a puzzlelike example. 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, the old gentleman says, there are three 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.

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 each diet rule with a boolean expression. The conjunction of the expressions describing the rules

$$(\neg b \to f) \ \wedge \ (b \wedge f \to \neg i) \ \wedge \ (i \vee \neg b \to \neg f) \tag{11}$$

yields a description of the diet. The boolean expression describing the diet evaluates to true if and only if the diet is observed by a meal. We can say that the expresion describes a boolean function that for the inputs b, f, i determines whether the diet is observed.

The expression (11) is only one possible description of the diet. Every equivalent boolean expression describes the diet as well, possibly in a way that is more enlightening than the rules given by the old gentleman. One possibility is the expression

$$b \wedge \neg(f \wedge i) \tag{12}$$

which says that a meal must come with beer but must avoid the combination of fish and ice cream. It is not difficult to verify that the expressions (11) and (12) are equivalent and hence describe the same boolean function.

We can check the equivalence of (11) and (12) with Coq. There are two different methods for doing so known as shallow and deep embedding.

With **shallow embedding**, the expressions (11) and (12) are represented as boolean Coq expressions and the equivalence is modelled as a universally quantified equality:

$$\forall\ b\ f\ i : \mathbb{B}.\ ((\neg b \to f) \ \wedge \ (b \wedge f \to \neg i) \ \wedge \ (i \vee \neg b \to \neg f)) \ = \ (b \wedge \neg(f \wedge i))$$

The proof proceeds by case analysis on the boolean variables b, f, and i, which yields 8 boolean equations that hold by definitional equivalence. The boolean connectives may be defined as shown in Section 1.

With **deep embedding**, the expressions (11) and (12) are represented with values of an inductive type Exp and the equivalence is modelled as boolean equivalence:

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

This time the variables are not quantified since they are represented as concrete numbers (e.g., as 0, 1, 2). The connectives are represented as abbreviations for conditionals. The proof now proceeds by introducing the universally quantified assignment α from the definition of $s \approx t$, simplification, and case analysis over the boolean expressions αb, αf, and αi, which eliminates the remaining occurrences of the assignment α.

If we consider the diet problem and the goal is just to establish the equivalence of the expressions (11) and (12), shallow embedding is the method of choice. Everything is clear and elegant. In this situation, deep embedding would just introduce unnecessary bureaucratic overhead.

If, on the other hand, we want to verify something like a SAT solver, deep embedding is a must since boolean expressions must be represented as data. It is important to understand how the transition from shallow embedding to deep embedding is done. Quantification over the boolean Coq variables b, g, and i in the diet example is replaced by quantification over a single assignment α, and the problem variables b, g, and i are represented as numbers.

Exercise 27 Show the equivalence of the expressions (11) and (12) in Coq. Explore both shallow and deep embedding and make sure you understand the many details and the relationship between the two approaches.

Exercise 28 Four girls agree on some rules for a party:

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

ii) Whoever does not dance with Richard 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.

Describe each rule with a boolean expression. Use the variables c (Chris), p (Peter), m (Michael), and r (Richard). Find a simple expression that is equivalent to the conjunction of the rules.

Remarks

1. Equivalence of boolean expressions may be seen as an abstract equality for expressions that is compatible with the syntactic structure of expressions (Exercise 22). Technically, equivalence of boolean expressions is obtained as universally quantified boolean equality (see the definition of equivalence). Coq supports abstract equalities with setoid rewriting (see Coq development).
2. Boolean expressions carry information. For many applications, only the information preserved by equivalence is relevant. The information represented by an expression and preserved by equivalence may be thought of as the set of assignments satisfying the expression, or as the boolean function described by the expression.
3. Treating syntax as data is standard in Computer Science. Accommodating syntax as data 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.

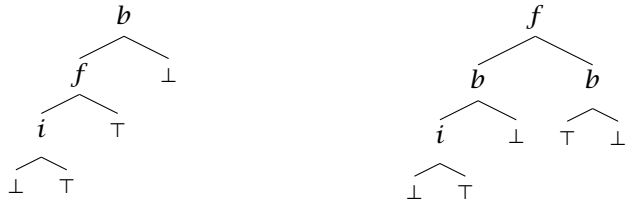
5 Decision Trees

A **decision tree** is a binary tree whose inner nodes are labelled with variables and whose leaves are labelled with $\bot$ or $\top$. Decision trees can be seen as boolean expressions. For this a compound tree T is represented as an expression $Cxuv$ where x is the root variable of T, u represents the left subtree of T, and v represents the right subtree of T. We will identify decision trees with their representation as boolean expressions.

Here are two decision trees:

Both trees are equivalent to the expression $x \wedge y$. Note that an assignment α satisfies a decision tree if it validates a path leading from the root to a leaf labelled with $\top$. At a node labelled with x, the path validated by α goes to the left if $\alpha x = \top$, and to the right if $\alpha x = \bot$.

Here are two decision trees that are equivalent to the expression $b \wedge \neg(f \wedge i)$ from the diet example:



A decision tree is called

- **consistent** if no variable appears twice on a path from the root to a leaf.
- **ordered** if the variables on every path from the root to a leaf appear in strictly ascending order (recall that the variables are numbers).
- **reduced** if for no inner node the left and the right subtree are identical.
- **prime** if it is reduced and ordered.

All trees shown above are consistent and reduced. Note that every ordered tree is consistent. If we assume $x < y$ and $b < f < i$, two of the four decision trees shown above are prime.

6 Prime Tree Normal Form

It turns out that every boolean expression is equivalent to a prime tree, and that two prime trees are equivalent if and only if they are identical. Thus prime trees provide

a unique normal for boolean expressions. We will exploit this fact and construct a function η that maps every expression to an equivalent prime tree. Since the prime tree normal form is unique, we will have $s \approx t \leftrightarrow \eta s = \eta t$ for all expressions s and t.[1]

We define a class of **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., $x < y$).

The definition of prime expressions agrees with the informal definition of prime trees. For proofs, the inductive definition of prime expressions will be used since it provides a helpful induction principle. The informal definition of prime trees was given so that we can use the language and intuitions coming with the tree metaphor.

Theorem 29 (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 $\mathsf{C}xuv$. ∎

Corollary 30 Prime expressions are equivalent if and only if they are identical.

The proof of the separation theorem omits details that need to be filled in a formal development. Some of the details are spelled out by the following facts.

Fact 31 Let $Cxst$ and $Cyuv$ be prime expressions. Then either $x \notin \mathcal{V}(Cyuv)$ or $x = y$ or $y \notin \mathcal{V}(Cxst)$.

Fact 32

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 $x \notin \mathcal{V}t + \mathcal{V}u$.

[1] Since equality of expressions is decidable, the equivalence $s \approx t \leftrightarrow \eta s = \eta t$ provides a second proof for the decidability of $s \approx t$.

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 construct a function η that maps every boolean expression to an equivalent prime tree. It turns out that the function solve we developed for the SAT solver can be modified such that it returns a prime tree. We start with the function

$$\begin{array}{lll} \eta'\ A\ s & := & \mathsf{match}\ A\ \mathsf{with} \\ & & \mid \mathsf{nil} \implies \mathsf{if}\ \tau s\ \mathsf{then}\ \top\ \mathsf{else}\ \bot \\ & & \mid x :: A \implies \mathsf{let}\ u := \eta' A(s^x_\top),\ v := \eta' A(s^x_\bot)\ \mathsf{in} \\ & & \qquad \mathsf{if}\ u = v\ \mathsf{then}\ u\ \mathsf{else}\ Cxuv \end{array}$$

By construction, η' yields a reduced decision tree. Moreover, if $\mathcal{V}s \subseteq A$, then $\eta' As$ yields a decision tree equivalent to s due to the following facts:

1. If s is ground, then $s \approx \top$ if $\alpha s = \top$ and $s \approx \bot$ if $\alpha s = \bot$.
2. $s \approx Cx(s^x_\top)(s^x_\bot)$ (expansion)
3. $Cxuu \approx u$ (reduction)

It is easy to see that η' yields an ordered decision trees if A is a strictly sorted list of variables. Thus

$$\eta s\ :=\ \eta'\ (\mathrm{sort}\ (\mathcal{V}s))\ s$$

provides the normalisation function we are looking for.

Lemma 33 $\eta' A\, s \approx s$ if $\mathcal{V}s \subseteq A$.

Lemma 34 $\mathcal{V}(\eta' A\, s) \subseteq A$.

Lemma 35 $\eta' A\, s$ is prime if A is strictly sorted.

Theorem 36 (Correctness) ηs is prime, $\eta s \approx s$, and $\mathcal{V}(\eta s) \subseteq \mathcal{V}s$.

The following facts follow from the correctness theorem and Corollary 30.

Corollary 37 $s \approx t \leftrightarrow \eta s = \eta t$.

Corollary 38 s is prime iff $\eta s = s$.

Corollary 39 (Idempotence) $\eta(\eta s) = \eta s$

Corollary 40 If $s \approx t$ and t is prime, then $\eta s = t$.

The following facts follow from the two theorems of this section without making use of the facts from Sections 2 and 3.

Corollary 41 s is unsatisfiable if and only if $\eta s = \bot$.

Corollary 42 Satisfiability of boolean expressions is decidable.

Corollary 43 s is satisfiable iff $\eta s \neq \bot$.

Exercise 44 Prove the following identities: $\eta\bot = \bot$, $\eta\top = \top$, $\eta x = \mathsf{C}x\top\bot$, and $\eta(\neg x) = \mathsf{C}x\bot\top$.

Exercise 45 Prove Corollaries 41, 42, and 43 without making use of the results from Sections 2 and 3.

Exercise 46 (SAT Solver) Construct a function that for every prime tree $s \neq \bot$ yields a satisfying assignment. Combine this function with η to obtain a SAT solver. Prove the correctness of your constructions.

Exercise 47 (Validity) Prove that η is valid iff $\eta s = \top$. Take the definition of validity from Exercise 26 but otherwise do not make use of the results from Sections 2 and 3.

Exercise 48 (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. Hint: Prove the more informative claim that there is a separating element that is one of the lists but not in the other.

Remarks

1. Prime trees are the tree version of a unique normal form Bryant [3] devised for BDDs (binary decision diagrams). BDDs are a graph representation of decision trees where identical subtrees can be represented with a single subgraph.
2. We can see the prime tree for an expression s as the semantic object **denoted** by s. Under this view, boolean equivalence is **denotational** in that two expressions are equivalent if and only if they denote the same prime tree.

7 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.

As an example consider the expression $x \vee (y \wedge \neg y)$. While x is significant for the expression, y is not. In fact, no variable but x is significant for the expression.

Fact 49 Every significant variable of an expression occurs in the expression.

Fact 50 Significant variables are stable under semantic equivalence.

Fact 51 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 $\alpha(u^x_\top) \neq \alpha(v^x_\top)$. 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 $s^x_\bot$. ∎

Fact 52 The significant variables of an expression are exactly the variables that appear in the prime tree of the expression.

Exercise 53 Write a function that for an expression yields a list containing exactly the significant variables of the expression. Prove the correctness of the function.

8 Boolean Algebra

Boolean algebra is a mathematical theory centered around the notion of a boolean algebra. A **boolean algebra** consists of a set X 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 1. The set X is called the **carrier** of the algebra. By convention, the constants $\top$ and $\bot$ are referred to as operations although they don't take arguments.

The standard example of a boolean algebra is the **two-valued boolean algebra** obtained with the type $\mathbb{B}$ and the operations introduced in Section 1.

An important class of boolean algebras are the power set algebras. The **power set algebra** $\mathcal{P}_X$ for a set X takes the power set $\mathcal{P}X$ as carrier, $\emptyset$ as $\bot$, X as $\top$, set

$$x \wedge y = y \wedge x \qquad x \vee y = y \vee x$$
$$x \wedge \top = x \qquad x \vee \bot = x$$
$$x \wedge \neg x = \bot \qquad x \vee \neg x = \top$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \qquad x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

Axioms are referred to as **commutativity**, **identity**, **negation**, and **distribution**.

Figure 1: Axioms for boolean algebras

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 every 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. It is known that every boolean algebra is isomorphic to a subalgebra of a power set algebra (Stone's representation theorem).

A main interest in boolean algebra is to understand which equations one can prove or disprove for every boolean algebra. For instance, one can prove that conjunction and disjunction are associative and satisfy the de Morgan laws ($\neg(x \wedge y) = \neg x \vee \neg y$ and $\neg(x \vee y) = \neg x \wedge \neg y$). One can also prove that $\top$ and $\bot$ are different if the algebra has at least two elements. As it turns out, the proofs of associativity and de Morgan are very tricky.

To prove equations in boolean algebra, one may assume a type X, the operations $\top$, $\bot$, $\neg$, $\wedge$, and $\vee$, and the 8 equational axioms. This amounts to a shallow embedding of boolean algebra in type theory. Try to prove $\neg\top = \bot$ and $\top \neq \bot$ if you are looking for a challenge (we will give away the tricks in the next section). To prove $\top \neq \bot$ you need to assume that X has at least two different elements.

There is a striking symmetry present in the axioms of boolean algebra known as **duality**. In Figure 1, the axioms on the right can be obtained from the axioms on the left by swapping $\top$ and $\bot$ and $\wedge$ and $\vee$. Swapping also produces the axioms on the left from the axioms on the right. One speaks of **dualisation**. An important fact about boolean algebra says that an equation holds in a boolean algebra if and only if the equation obtained by dualisation holds in the boolean algebra. A rigorous proof of this fact requires a deep embedding where expressions appear as data.

There are different axiomatizations of boolean algebras in the literature [5]. Our axiomatization is a variant of the axiomatization used in Whitesitt's [6] textbook, which in turn is a variant of an axiomatization devised by Huntington [4]. Huntington [4] discovered that associativity, which until then appeared as an axiom, can be derived from the other axioms. Boolean algebra originated from the work of Boole [2]. Boole thought of conjunction as class intersection and of disjunction as

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

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

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

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

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

Figure 2: Axiomatic equivalence of boolean expressions

union of disjoint classes [1].

9 Axiomatic Equivalence

We shall show that an equation holds in a boolean algebra if and only if it holds in the two-valued boolean algebra. This remarkable result makes it possible to transfer the methods for two-valued boolean logic to general boolean algebras. To show this result, we need a deep embedding of boolean algebra into type theory where expressions appear as data.

We start with an inductive type Exp providing **boolean expressions**

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

obtained with $\top$, $\bot$, variables, negation, conjunction, and disjunction. We model the axioms of boolean algebra and the rules of equational reasoning with an inductive predicate $s \equiv t$ called **axiomatic equivalence**. The definition of axiomatic equivalence appears in Figure 2. Here are some explanations and remarks:

1. The axioms for boolean algebras are expressed with premise-free rules (written without a line). The rules quantify over expressions s, t, and u. In contrast, the axioms quantify over values of the boolean algebra in the shallow embedding.
2. The rules providing for equational reasoning are **symmetry**, **transitivity**, and **compatibility** with $\neg$, $\wedge$, and $\vee$. The rule for reflexivity is omitted since it can be derived with the other rules. The compatibility rules providing for rewriting of the right argument of conjunctions and disjunctions are omitted since they can be derived with the commutativity axioms.
3. Rewriting is the basic mode of equational reasoning. In Coq, the rules providing equational reasoning for $s \equiv t$ may be registered with setoid rewriting so that one can rewrite with equivalences.

4. The fact that a boolean algebra has at least two elements is not expressed in the definition of axiomatic equivalence. It will be rarely needed.
5. The inductive definition of $s \equiv t$ should be seen as a proof system for judgements $s \equiv t$. Only very few proofs will be carried out in the basic proof system. Most proofs will make use of setoid rewriting and derived rules.
6. Some important proofs will use induction on the inductive predicate $s \equiv t$. These inductions have to consider one proof obligation for every rule defining $s \equiv t$. For this reason it is convenient to have as few defining rules as possible. This is the main argument for not having reflexivity as a defining rule.

Fact 54 (Reflexivity) $s \equiv s$.

Proof With identity and symmetry we have $s \wedge \top \equiv s$ and $s \equiv s \wedge \top$. Thus $s \equiv s$ with transitivity. ∎

We now prove the dualisation theorem for boolean algebra. The proof is not difficult. We start with the definition of 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 55 (Involution) $\hat{\hat{s}} = s$.

Theorem 56 (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

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

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 very helpful in the following proofs.

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 so that the accompanying induction principle has

few cases. Given that when constructing 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 57 (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 $\bot \equiv s \wedge \neg s \equiv s \wedge (\neg s \vee \bot)$. ∎

Fact 58 (Exfalso) If $\top \equiv \bot$, then $s \equiv t$ for all expressions s and t.

Proof Follows with identity and annulation. ∎

Fact 59

$s \wedge s \equiv s$	$s \vee s \equiv s$	**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 60 (Expansion) $s \equiv (t \vee s) \wedge (\neg t \vee s)$ and $s \equiv (t \wedge s) \vee (\neg t \wedge s)$.

Fact 61 (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 62 (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 identity (both sides reduce to $(\neg s \vee t) \wedge (\neg s \vee u)$). ∎

Fact 63 (Reduction to $\bot$) $s \equiv t$ iff $s \wedge \neg t \vee \neg s \wedge t \equiv \bot$.

Fact 64 (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 65 (Double Negation) $\neg\neg s \equiv s$.

Proof Follows with uniqueness of complements, commutativity, and negation. ∎

Fact 66 (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 67 Derive the compatibility rules for the right arguments of conjunctions and disjunctions:

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

Exercise 68 Prove that $s \equiv t$ iff $\neg s \equiv \neg t$.

Exercise 69 Our axioms for boolean algebras (see Figure 1) 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.

10 Soundness of Axiomatic Equivalence

As before, an assignment α is a function mapping variables to booleans. Given an assignment α, we can **evaluate** an expression s to a boolean αs. Evaluation of negations, conjunctions, and disjunctions follows their standard boolean semantics (Section 1). We now define **assignment equivalence** for the expressions of boolean algebra following the scheme we know from boolean expressions (Section 3):

$$s \approx t \;:=\; \forall \alpha.\ \alpha s = \alpha t$$

We can now prove that axiomatic equivalence entails assignment equivalence.

Fact 70 (Soundness) If $s \equiv t$, then $s \approx t$.

Proof By induction on $s \equiv t$. ∎

The proof is straightforward. Intuitively, soundness holds since assignment equivalence satisfies the axioms of boolean algebra. That assignment equivalence satisfies symmetry, transitivity, and the compatibility rules certifies its design as an abstract equality.

Corollary 71 (Consistency) $\top \not\equiv \bot$.

Proof Follows from soundness since $\alpha\top \neq \alpha\bot$ for every assignment α. ∎

Theorem 72 (Ground Evaluation) Let s be a ground expression and α be an assignment. Then $s \equiv \top$ if $\alpha s = \top$ and $s \equiv \bot$ if $\alpha s = \bot$.

Proof By induction on s using the evaluation laws (Fact 57). ∎

Corollary 73 If s is ground, then $s \equiv \top$ or $s \equiv \bot$.

As before, we call two expressions s and t **separable** if there exists an assignment α such that $\alpha s \neq \alpha t$.

Exercise 74 Prove $x \not\equiv y$ and $s \not\equiv \neg s$.

Exercise 75 Prove or disprove the following propositions:

a) $\forall st.\ s \vee t \equiv \bot \leftrightarrow s \equiv \bot \wedge t \equiv \bot$

b) $\forall st.\ s \equiv t \leftrightarrow s \wedge \neg t \vee \neg s \wedge t \equiv \bot$

c) $\forall s.\ s \equiv \top \leftrightarrow \neg s \equiv \bot$

d) $\forall st.\ s \vee t \equiv \bot \leftrightarrow s \equiv \bot \vee t \equiv \bot$

e) $\forall st.\ \neg(s \equiv t) \leftrightarrow s \equiv \neg t$

11 Completeness of Axiomatic Equivalence

We will now show that assignment equivalence entails axiomatic equivalence. We refer to this result as **completeness**. Together with soundness (Fact 70), completeness says that axiomatic equivalence agrees with assignment equivalence. The following lemma formulates the idea for the completeness proof.

Lemma 76
Suppose there is a function $\forall s.\ \{\alpha \mid \alpha s = \top\} + \{s \equiv \bot\}$. Then $s \equiv t$ if $s \approx t$.

Proof Let $s \approx t$. By the assumption the expression $u := s \wedge \neg t \vee \neg s \wedge t$ is either satisfiable or equivalent to $\bot$. Satisfiability of u contradicts the assumption $s \approx t$. If $u \equiv \bot$, then $s \equiv t$ follows with Fact 63. ∎

The SAT solver from Section 2 gives us a function $\forall s.\ \{\alpha \mid \alpha s = \top\} + \{\neg\mathrm{sat}\ s\}$ (Theorem 6). We can obtain a function as required by Lemma 76 by adapting the solver to the expressions of boolean algebra and by strengthening the correctness proof to $s \equiv \bot$. For the strengthening of the correctness proof we need an expansion theorem for axiomatic equivalence. The definition of the substitution operation s^x_t needed for the formulation of the expansion property is routine.

Theorem 77 (Expansion)

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 The first two claims are lemmas needed for the proof of the third claim, which formulates the expansion property.

The first claim follows by induction on s. The base case for variables follows with $x \wedge x \equiv x \wedge \top$. The inductive cases follow by rewriting with the equivalences

- $x \wedge \neg s \equiv x \wedge \neg(x \wedge s)$
- $x \wedge (s \wedge t) \equiv (x \wedge s) \wedge (x \wedge t)$
- $x \wedge (s \vee t) \equiv (x \wedge s) \vee (x \wedge t)$

One first rewrites with the appropriate equivalence to push x below the operation, then rewrites with the inductive hypotheses, and finally pushes x up again by rewriting with the first equivalence in reverse direction. Here is the rewrite chain for negation: $x \wedge \neg s \equiv x \wedge \neg(x \wedge s) \equiv x \wedge \neg(x \wedge s^x_\top) \equiv x \wedge \neg(s^x_\top) \equiv x \wedge (\neg s)^x_\top$.

The proof of the second claim is analogous to the proof of the first claim.

The third claim is a straightforward consequence of the first and second claim: $s \equiv s \wedge (x \vee \neg x) \equiv x \wedge s \vee \neg x \wedge s \equiv x \wedge s^x_\top \vee \neg x \wedge s^x_\bot$. ∎

Lemma 78 There is a function $\forall s.\ \{\,\alpha \mid \alpha s = \top\,\} + \{s \equiv \bot\}$.

Proof By strengthening the correctness proof of the SAT solver from Section 2. For this ground evaluation (Theorem 72) and expansion (Theorem 77) are essential. ∎

Theorem 79 (Agreement) $s \equiv t \leftrightarrow s \approx t$.

Proof Follows with Fact 70 and Lemmas 76 and 78. ∎

Theorem 80 (Decidability) Axiomatic equivalence $s \equiv t$ is decidable.

Proof Follows with Facts 63, 70, and Lemma 78. ∎

Exercise 81 (Coq project) Define and verify a prime tree normaliser for axiomatic equivalence. Use the notation $Cxst := x \wedge s \vee \neg x \wedge t$ for conditionals. First define prime trees and show the separation theorem. Then define a normaliser η and verify $\eta s \equiv s$ and ηs is prime.

12 Substitutivity

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

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.

$$\tilde{\theta}\top := \top \qquad \tilde{\theta}x := \theta x \qquad \tilde{\theta}(s \wedge t) := \tilde{\theta}s \wedge \tilde{\theta}t$$
$$\tilde{\theta}\bot := \bot \qquad \tilde{\theta}(\neg s) := \neg(\tilde{\theta}s) \qquad \tilde{\theta}(s \vee t) := \tilde{\theta}s \vee \tilde{\theta}t$$

Fact 82 (Substitutivity) If $s \equiv t$, then $\widetilde{\theta}s \equiv \widetilde{\theta}t$.

Proof By induction on the derivation of $s \equiv t$. The proof is straightforward since the rules of the proof system are closed under substitution. ∎

Since axiomatic equivalence agrees with assignment equivalence, substitutivity also holds for assignment equivalence.

Fact 83 (Substitutivity) If $s \approx t$, then $\widetilde{\theta}s \approx \widetilde{\theta}t$.

Theorem 84 (Negative Completeness)
$s \not\equiv t \leftrightarrow \exists\theta.\ \widetilde{\theta}s \equiv \top \wedge \widetilde{\theta}t \equiv \bot \vee \widetilde{\theta}s \equiv \bot \wedge \widetilde{\theta}t \equiv \top$.

Proof The direction from right to left follows with substitutivity and consistency (Corollary 71). For the other direction assume $s \not\equiv t$. Then $s \not\approx t$ by agreement (Theorem 79). By Fact 18 there is an assignment α such that $\alpha s \neq \alpha t$. The claim follows by taking for θ the substitution corresponding to α. ∎

13 Generalised Assignment Equivalence

Let X be the carrier of a boolean algebra. We can consider a generalised assignment $\alpha : \mathrm{Var} \to X$ and evaluate an expression s into an element of X. This gives us a generalised notion of assignment equivalence $s \approx_X t$. We may ask whether assignment equivalence $s \approx_X t$ agrees with axiomatic equivalence $s \equiv t$. The soundness direction is as easy as in the boolean case.

Fact 85 (Soundness) If $s \equiv s$, then $s \approx_X t$.

The completeness direction only holds if X has at least two elements. If X has two elements, we can show that $\top$ and $\bot$ are different in X. This means that a boolean assignment separating s and t yields an assignment into X separating s and t in X. This is all we need for the completeness direction.

Theorem 86 (Generalised Agreement) Let X be a boolean algebra with at least two elements. Then $s \equiv t$ if $s \approx_X t$.

Exercise 87 (Independence of consistency axiom) Give an algebra satisfying all axioms for boolean algebras but $\top \neq \bot$.

Exercise 88 (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 89 (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.

Exercise 90 Prove that assignment equivalence satisfies substitutivity without using axiomatic equivalence.

14 Ideas for Next Version

- Move the diet example to Section 1. One can show the equivalence of different boolean codings of the diet without explicit syntax. Use the idea of a tool that simplifies expressions to motivate explicit syntax and assignment equivalence.
- Define expressions and assignment equivalence before developing the SAT solver. Equivalence checking and computation of separating assignments reduce to SAT solving, which is the basic operational service.
- Present tableau decomposition (lists of signed formulas, no substitution) as a second technique for SAT solving. The tableau method may simplify the completeness proof for axiomatic equivalence since axiomatic expansion is not needed.
- Make more explicit that SAT solving and prime normalisation work for different syntactic systems (outlined in Exercise 2). The system with just variables and nand works with tableau decomposition but does not work with variable elimination (add $\top$ or $\bot$).
- Maybe do a few proofs for an assumed boolean algebra do see the difference to axiomatic equivalence.

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

[5] 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.

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