

Theory of Automated Reasoning
An Introduction

Antti-Juhani Kaijanaho

Intended as compulsory reading for the Spring 2004 course on
Automated Reasoning at Department of Mathematical
Information Technology, University of Jyväskylä.

CHAPTER 3

The Ground Case

1. Sentential language	23
2. From ground to sentential formulae	26
3. Clausal forms	28
4. Inference in the sentential language	31
4.1. Semantic tableaux	31
4.2. Davis–Putnam procedure	34
4.3. Sentential resolution	36
4.4. On heuristics	36

1. Sentential language

METADefinition 3.1 (Sentential language). Let V be a first-order vocabulary containing at least one constant symbol and at least one predicate symbol and let p be a well-formed ground formula in the first-order language over V . Now, let q_1, \dots, q_n be a permutation of the atomic subformulae of p and let x_1, \dots, x_n be a n -permutation (permutation of any n elements) of \mathcal{V} . We rewrite p by replacing in it each atomic subformula q_i with the corresponding variable x_i . The set of results of this rewrite for every possible such permutations is denoted P_p . We now call the set $\mathcal{F} = \{q \in P_p \mid p \in \mathcal{F}_V\} \cup \{\top, \perp\}$ a *sentential language* and its members (*well-formed*) *sentential formulae*. When used in this context, the elements of the set \mathcal{V} are called *sentential variables*.

For sentential formulae p and q , we denote by $p[q/x]$ the formula obtained from p by replacing all occurrences of the variable x with q .

It is important to understand the difference in function between a variable in a first-order language and a sentential variable. In first-order languages, variables are placeholders for terms, but in sentential languages, variables are placeholders for formulae.

The formula \top is pronounced *verum* and the formula \perp *falsum*. They are *sentential constants*.

METAPROPOSITION 3.2. *All sentential languages are identical: if L and L' are sentential languages, then $L = L'$ holds.*

PROOF. Left as an exercise. □

Since all sentential languages are identical, it makes sense to think that there is only one and speak of *the* sentential language. The sentential language has many names: popular names are *propositional language*, *propositional calculus* and *propositional logic*. Authors who use these names also tend to call sentential formulae *propositions* and sentential variables *propositional variables*. The name “sentential language” refers to its apparent usefulness in analysing the structure of complex declarative *sentences* in natural languages.

METADefinition 3.3 (Semantics of the sentential language). A *general truth valuation* is a function f from the sentential language to $\{0, 1\}$ obeying the following equations of informal arithmetic (where p and q are sentential formulae):

$$(30) \quad f(\top) = 1$$

$$(31) \quad f(\perp) = 0$$

$$(32) \quad f(\neg p) = 1 - f(p)$$

$$(33) \quad f(p \wedge q) = f(p)f(q)$$

$$(34) \quad f(p \vee q) = f(p) + f(q) - f(p)f(q)$$

$$(35) \quad f(p \rightarrow q) = 1 - f(p) + f(q)$$

$$(36) \quad f(p \leftarrow q) = 1 - f(q) + f(p)$$

$$(37) \quad f(p \leftrightarrow q) = f(p)f(q) + (1 - f(p))(1 - f(q))$$

$$(38) \quad f(p \mid q) = 1 - f(p)f(q)$$

$$(39) \quad f(p \downarrow q) = (1 - f(p))(1 - f(q))$$

$$(40) \quad f(p \vee\vee q) = f(p) + f(q) - 2f(p)f(q)$$

Let ϕ be a truth valuation and p be a sentential formula. We define $\phi \models p$ as $\phi(p) = 1$ and $\phi \not\models p$ as $\phi(p) = 0$.

We say that p is *true* for ϕ , when $\phi \models p$ holds, and that p is *false* for ϕ , when $\phi \not\models p$ holds. Further, we say that p is a *tautology*, denoted $\models p$, if it is true for all possible truth valuations, *satisfiable*, if it is true for some (perhaps all) possible truth valuations, *unsatisfiable* if it is true for no possible truth valuation and *contingent* if it is satisfiable but not a tautology. Two sentential formulae, p and q , are *logically equivalent* if $p \leftrightarrow q$ is a tautology.

METAPROPOSITION 3.4. *Every tautology is logically equivalent to \top . Every unsatisfiable formula is logically equivalent to \perp .*

PROOF. Trivial. □

A *truth table* is essentially a tabular presentation of the above definition. We can develop a truth table (Table 1) for each connective by observing how the truth of each one-connective sentential formula behaves with respect to the truth of its sentential variables.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftarrow q$	$p \leftrightarrow q$	$p \mid q$	$p \downarrow q$	$p \vee\vee q$
1	1	0	1	1	1	1	1	0	0	0
1	0	0	0	1	0	1	0	1	0	1
0	1	1	0	1	1	0	0	1	0	1
0	0	1	0	0	1	1	1	1	1	0

TABLE 1. The basic truth table

Truth tables present us with a simple but expensive algorithm for checking for satisfiability and tautologiness. But first, let us define some convenient notation.

METADEFINITION 3.5. Let P be a nonempty finite set of sentential formulae. Now, we define $\bigwedge P$ as $f(1) \wedge \cdots \wedge f(n)$ and $\bigvee P$ as $f(1) \vee \cdots \vee f(n)$, where n is the size of P and f is any injection from $\{1, \dots, n\}$ to P . We further define $\bigwedge \emptyset$ as \perp and $\bigvee \emptyset$ as \top .

METAPROPOSITION 3.6. *The new operators \bigwedge and \bigvee are well-defined; that is, regardless of how f is chosen, the result is the same.*

PROOF. Follows from associativity and commutativity of conjunction and disjunction. \square

METAPROPOSITION 3.7. *Let P be a finite set of sentential formulae and let p be a sentential formula not in P . Then, both*

$$\bigwedge (P \cup \{p\}) \equiv p \wedge \bigwedge P$$

and

$$\bigvee (P \cup \{p\}) \equiv p \vee \bigvee P$$

hold.

PROOF. Omitted. \square

Now, we can give the algorithm.

ALGORITHM 3.8. Let input be a finite set of sentential formulae P and a sentential formula q . Let p be $\bigwedge P \rightarrow q$ (if P is empty, p is q). Now, let r_1, \dots, r_n be a permutation of all distinct subformulae of p where formulae are preceded by their subformulae. Let k be the number of distinct atomic subformulae of p . Allocate a two-dimensional array with 2^k rows and n columns, both indexed starting from 1. Do the following for each $i = 1, \dots, n$ (in ascending order of i).

If $i \leq k$ holds, fill the column i first with 2^{i-1} ones, then with 2^{i-1} zeros, then with 2^{i-1} ones and so on until the whole column is filled. Otherwise, for each $j = 1, \dots, 2^k$, look up the cells at row j , column as corresponding to each of the immediate subformulas of the formula r_i , and use them to look up the correct row in Table 1, then copy the

value from that row at the column corresponding to the formula r_i to the array at row j , column i .

Finally, check if the n th column of the array contains any ones. If not, output *unsatisfiable*. Otherwise, check if it contains any zeros. If not, output *tautology*, otherwise output *contingent*.

METAPROPOSITION 3.9. *Algorithm 3.8 is correct (i.e. terminates normally for all valid inputs and produces the correct result whenever it terminates normally).*

PROOF. Omitted. □

2. From ground to sentential formulae

Recall that a *ground term* is a term which contains no variables, bound or free, and no ϵ -terms. Likewise, a *ground formula* contains no variables, bound or free, and no quantifiers. Ground terms can generally be ignored, since they are effectively constants. Ground formulae are more interesting, as can be seen from the following metadefinitions and metapropositions:

METADefinition 3.10. Formulae of the form $(\exists, r, t_1, \dots, t_{a(r)})$ (in other words, $r(t_1, \dots, t_{a(r)})$) are called *atomic formulae*. If $p \in f(q)$ holds for the f of Definition 2.12, then p is a *subformula* of q .

METAPROPOSITION 3.11. *All formulae contain at least one atomic subformula.*

PROOF. This is evident from the definition of a formula (atomic formulae form the base case of the recursion). Note that a formula is a subformula of itself, and hence this holds also for atomic formulae. □

METALEMMA 3.12. *For every well-formed ground formula p , its extension $\llbracket p \rrbracket_M$ in any model M is either \mathcal{U} or \emptyset .*

PROOF. Evident from the fact that no variables are present in ground formulae. □

The following proposition is the key to combining ground formulae and sentential formulae.

METAPROPOSITION 3.13 (Truth-functionality). *A well-formed ground formula is a truth function of its atomic subformulae: for each well-formed ground formula p with its atomic subformulae being q_1, \dots, q_n (possibly containing duplicates), there is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ such that for every model M , the assertion $M \models p$ holds if and only if $f(b_1, \dots, b_n) = 1$ holds, where each b_i is 1 if and only if $M \models q_i$ holds, and 0 otherwise.*

The function f is called the *truth function* of the formula and the b_i are called the *truth values* of the atomic subformulae.

PROOF. We proceed by structural induction.

- (1) Let p be atomic. Then clearly $n = 1$ and $q_i = p$ hold. The required function f is thus the identity function in $\{0, 1\}$. Other possible functions are obtained by defining $f(b_1, \dots, b_n) = f(b_1)$, where $b_i = b_j$ holds.
- (2) Let p be $\neg q$ with b_1, \dots, b_n being the truth values atomic subformulae of q . Note that they are also the atomic formulae of p . Let M be a model where $M \models q$ holds. By definition $\llbracket q \rrbracket_M = \mathcal{U}$ holds. $\llbracket \neg q \rrbracket_M = \emptyset$ holds, and so $M \not\models \neg q$ holds. Let M be a model where $M \not\models q$ holds. By definition and lemma 3.12, $\llbracket q \rrbracket_M \neq \mathcal{U}$ holds. Now, therefore, $\llbracket \neg q \rrbracket_M = \emptyset$. A suitable f for p is, therefore,

$$(b_1, \dots, b_n) \mapsto \begin{cases} 1 & \text{if } g(b_1, \dots, b_n) = 0 \\ 0 & \text{if } g(b_1, \dots, b_n) = 1 \end{cases}$$

where g is a truth function for q . Other suitable functions can be obtained by permuting the atomic subformulae, and by removing or adding duplicate atomic subformulae.

- (3) Let p be $q \wedge r$ with b_1, \dots, b_m being the atomic subformulae of q and b_{m+1}, \dots, b_n (where $m < n$ holds) being the atomic subformulae of r . There are four cases:
 - (a) Let M be a model where $M \models p$ and $M \models q$ hold. Then, $\llbracket q \wedge r \rrbracket_M = \llbracket q \rrbracket_M \cap \llbracket r \rrbracket_M = \mathcal{U} \cap \mathcal{U} = \mathcal{U}$ holds.
 - (b) Let M be a model where $M \models p$ and $M \not\models q$ hold. Then, $\llbracket q \wedge r \rrbracket_M = \llbracket q \rrbracket_M \cap \llbracket r \rrbracket_M = \mathcal{U} \cap \emptyset = \emptyset$ holds.
 - (c) Let M be a model where $M \not\models p$ and $M \not\models q$ hold. Then, $\llbracket q \wedge r \rrbracket_M = \llbracket q \rrbracket_M \cap \llbracket r \rrbracket_M = \emptyset \cap \mathcal{U} = \emptyset$ holds.
 - (d) Let M be a model where $M \not\models p$ and $M \models q$ hold. Then, $\llbracket q \wedge r \rrbracket_M = \llbracket q \rrbracket_M \cap \llbracket r \rrbracket_M = \emptyset \cap \emptyset = \emptyset$ holds.

Therefore, for p , a suitable f is

$$(b_1, \dots, b_n) \mapsto \begin{cases} 1 & \text{if } f_q(b_1, \dots, b_m) = 1 \text{ and } f_r(b_{m+1}, \dots, b_n) = 1 \text{ hold} \\ 0 & \text{otherwise} \end{cases}$$

where f_q and f_r are the truth functions of q and r , respectively. Other suitable functions can be obtained by permuting the atomic subformulae, and by removing or adding duplicate atomic subformulae.

- (4) Other cases are taken care of the above due to Metaproposition 2.22 case 3.

□

METACOROLLARY 3.14. *The above result holds even if some or all of q_i are replaced with non-atomic subformulae as long as every atomic subformula is a subformula of some q_i .*

PROOF. We can split the induction into two by introducing artificial base cases before encountering an atomic subformula, as long as we can reach every atomic subformula by continuing induction. \square

Now, the following proposition allow us to move freely between first-order ground formulae and sentential formulae:

METAPROPOSITION 3.15. *For every first-order structure M there is a truth valuation ϕ such that for every ground formula p and every sentential formula p' generated from p as in Definition 3.1, $M \models p$ holds whenever $\phi \models p'$ holds, and vice versa.*

PROOF. Left as an exercise. \square

METACOROLLARY 3.16. *There is a sound inference system that is complete and decidable for ground formulae (i.e. where any valid ground formula is a theorem and any refutable ground formula is a nontheorem).*

PROOF. Use the truth table method, translating ground formulae into sentential formulae at input and translating tautologiness into theoremhood and unsatisfiability and contingency into nontheoremhood. \square

3. Clausal forms

METADefinition 3.17 (Clausal forms). Any sentential formula that is a sentential variable or \top is a *positive literal*. Any sentential formula that is a negation of a sentential variable or \perp , is a *negative literal*. Positive literals and negative literals are all *literals*. The *complement* of a negative literal $\neg x$ or \perp is the corresponding positive literal x or \top ; conversely, the complement of a positive literal x or \top is the corresponding negative literal $\neg x$ or \perp . Two literals are *complementary* if they are each others' complements. A formula that is not a literal is a *nonliteral*.

A finite set of literals is called a *clause* and a *dual clause*.

A set of clauses C is a *clause form* of a sentential formula p if $p \equiv \bigwedge \bigvee C$ holds. A set of dual clauses C is a *dual clause form* of p if $p \equiv \bigvee \bigwedge C$ holds.

The reason for two names for the same thing is that they suggest different interpretations. A set of literals that is a clause is always interpreted as a disjunction of the literals, whereas a dual clause is always interpreted as a conjunction of its literals. Similarly, a set of clauses is interpreted as a conjunction of the disjunctions that the clauses are, and a set of dual clauses is interpreted as a disjunction of the conjunctions that the clauses are. This is also evident from the definitions of clause and dual clause forms.

We denote the complement of a literal ℓ by $\bar{\ell}$.

α	α_1	α_2	β	β_1	β_2
$\neg\neg p$	p	p			
$p \wedge q$	p	q	$\neg(p \wedge q)$	$\neg p$	$\neg q$
$\neg(p \vee q)$	$\neg p$	$\neg q$	$p \vee q$	p	q
$\neg(p \rightarrow q)$	p	$\neg q$	$p \rightarrow q$	$\neg p$	q
$\neg(p \leftarrow q)$	$\neg p$	q	$p \leftarrow q$	p	$\neg q$
$p \leftrightarrow q$	$p \leftarrow q$	$p \rightarrow q$	$\neg(p \leftrightarrow q)$	$\neg(p \leftarrow q)$	$\neg(p \rightarrow q)$
$\neg(p \mid q)$	p	q	$p \mid q$	$\neg p$	$\neg q$
$p \downarrow q$	$\neg p$	$\neg q$	$\neg(p \downarrow q)$	p	q
$\neg(p \vee\vee q)$	$p \leftarrow q$	$p \rightarrow q$	$p \vee\vee q$	$\neg(p \leftarrow q)$	$\neg(p \rightarrow q)$

TABLE 2. Lis–Smullyan uniform notation for sentential formulae

METATHEOREM 3.18. *Every sentential formula has both a clause form and a dual clause form.*

We will prove this theorem by giving algorithms that construct a clause form or a dual clause form of a sentential formula. To present these algorithms concisely, we need a uniform notation independently discovered by Lis and Smullyan in the 1960s. The basic idea of this notation comes from the observation that every nonliteral formula can be thought as essentially conjunctive or essentially disjunctive (in fact, as noted earlier, all sentential formulae can be written with just negation, conjunction and disjunction). Now, every nonliteral is categorized either as an α -formula (*essentially conjunctive formula*) or as a β -formula (*essentially disjunctive formula*), associating with each α -formula a pair of decomposition formulas α_1 and α_2 such that $\alpha \equiv \alpha_1 \wedge \alpha_2$ holds, and associating with each β -formula a pair of decomposition formulas β_1 and β_2 such that $\beta \equiv \beta_1 \vee \beta_2$ holds. In any formula, α denotes an arbitrarily chosen α -formula, and α_1 and α_2 are the α_1 and α_2 of that formula. Likewise, β denotes an arbitrarily chosen β -formula, and β_1 and β_2 are the β_1 and β_2 of that formula. Table 2 shows how formulae are categorized and decomposed.

ALGORITHM 3.19 (Clause form). Let the input be a sentential formula p and let r be $\{\{p\}\}$. Now, for as long as $\bigcup r$ contains at least one nonliteral, repeat the following:

- (1) Choose a nonliteral-containing member from r ; let it be c .
- (2) Choose a nonliteral from c .
- (3) If the chosen formula was $\neg\neg q$ for some q , update r to be $r' \cup \{c' \cup \{r\}\}$, where r' is $r \setminus \{c\}$ and c' is $c \setminus \{\alpha\}$.
- (4) If some other α -formula was chosen, update r to be $r' \cup \{c' \cup \{\alpha_1\}, c' \cup \{\alpha_2\}\}$, where r' is $r \setminus \{c\}$ and c' is $c \setminus \{\alpha\}$.

- (5) If a β -formula was chosen, update r to be $r' \cup \{c' \cup \{\beta_1, \beta_2\}\}$, where r' is $r \setminus \{c\}$ and c' is $c \setminus \{\beta\}$.

The output is the final update of r .

METAPROPOSITION 3.20. *Algorithm 3.19 terminates for all inputs regardless of how choices are made.*

PROOF. Let n be the number of nonliteral subformulae of every formula in $\bigcup r$. Clearly n is nonnegative at all times, and clearly the algorithm terminates if and only if $n = 0$ holds. Now, it remains to show that every iteration decreases n by at least one. We leave that as an exercise. \square

METALEMMA 3.21. *The output of Algorithm 3.19 is a set of clauses.*

PROOF. For simplicity, we assume that the initial state of r is the input, and that it may contain any number of sets of formulae as input. The actual algorithm is a special case of this.

We proceed by induction over the number n of nonliteral subformulae in formulae of $\bigcup r$. The base case, $n = 0$, is trivial: it can only happen if the input is a set of clauses already, in which case the output is a set of clauses. We now assume that the algorithm outputs a set of clauses if given as input a set of formula sets containing strictly less than k nonliteral subformulae. Consider input containing exactly k nonliteral subformulae. Clearly, $n = k$ holds at the beginning. Now, as noted in the previous proof, an iteration always decreases n ; we therefore let the algorithm run for one iteration; the proposition then follows from the induction hypothesis. \square

METAPROPOSITION 3.22. *The output of Algorithm 3.19 is a clause form of its input formula.*

PROOF. It suffices to show that $\bigwedge \bigvee r \equiv p$ is a loop invariant, that is, that it holds in the beginning and that each iteration preserves it. We leave this as an exercise. \square

ALGORITHM 3.23 (Dual clause form). Let p be a sentential formula and let r be $\{\{p\}\}$. Now, for as long as $\bigcup r$ contains at least one nonliteral, repeat the following:

- (1) Choose a nonliteral-containing member from r ; let it be c .
- (2) Choose a nonliteral from c .
- (3) If the chosen formula was $\neg\neg q$ for some q , update r to be $r' \cup \{c' \cup \{r\}\}$, where r' is $r \setminus \{c\}$ and c' is $c \setminus \{\alpha\}$.
- (4) If some other α -formula was chosen, update r to be $r' \cup \{c' \cup \{\alpha_1, \alpha_2\}\}$, where r' is $r \setminus \{c\}$ and c' is $c \setminus \{\alpha\}$.
- (5) If a β -formula was chosen, update r to be $r' \cup \{c' \cup \{\beta_1\}, c' \cup \{\beta_2\}\}$, where r' is $r \setminus \{c\}$ and c' is $c \setminus \{\beta\}$.

The output is the final update of r .

METAPROPOSITION 3.24. *Algorithm 3.23 is correct, that is, it terminates for all valid inputs and its output is a dual clause form of its input.*

PROOF. Proceeds similarly to the corresponding proofs for the clause algorithm above. \square

4. Inference in the sentential language

In this section, we will review several inference systems for the ground case, using the sentential language.

4.1. Semantic tableaux. Semantic tableaux are based on the ideas of Gerhard Gentzen from the 1930s, specifically his sequent calculi. The idea of tableaux was originally discovered by Jaakko Hintikka and E. W. Beth independently of each other in the 1950s, and was further refined by Raymond Smullyan in the 1960s (Smullyan called his version *analytic tableaux*). Tableaux as an automated reasoning method was popularized by Smullyan's student, Melvin Fitting, in the first edition of his textbook as well as by several successful tableau-based systems in the early 1990s. The presentation here is influenced by Ben-Aris presentation¹.

METADefinition 3.25 (Labelled trees). A *labelled tree* is a quintuple (E, L, r, p, l) for which the following hold:

- (1) $r \in E$.
- (2) $p : (E \setminus \{r\}) \rightarrow E$.
- (3) For every $e \in E$, there is a finite set $\{e_1, \dots, e_n\}$ such that $e = e_n$ holds, $e_1 = r$ holds, $p(e_{i+1}) = e_i$ holds for all $i = 1, \dots, n-1$, and $e_i = e_j$ implies $i = j$ for all $i = 1, \dots, n-1$. The size of that set, n , is the *depth* of the node e .
- (4) For no $e \in E$ is there a finite set $\{e_1, \dots, e_n\}$ such that $e = e_n = e_1$ holds and $p(e_i) = e_{i+1}$ holds for all $i = 1, \dots, n-1$.
- (5) $l : E \rightarrow L$.

The elements of E are called *nodes* or *vertices*, the elements of L are called *labels*, and the node r is called the *root*. For each node e , the node $s(e)$ is called its *parent*. The *children* of a node are all the nodes whose parent it is. Two nodes are *siblings* if they have a common parent. A node is a *leaf* if it has no children; all other nodes are *internal*. A sequence of nodes is a *branch* if its first node is the root, either it is infinite or its last node is a leaf, and each of its nodes is a parent of the next node. A labelled tree is *binary* if all its nodes have at most two children.

¹Mordechai Ben-Ari: *Mathematical Logic for Computer Science*, Second Edition, London, Springer, 2001.

METADefinition 3.26. A (*sentential*) *semantic tableau* is any labelled binary tree whose labels are sets of sentential formulae, satisfying the following criteria:

- (1) The root node is labelled with a singleton set.
- (2) If a node e has two children, then its label $l(e)$ contains a β -formula and the labels of the two children e_1 and e_2 are $l(e_1) = (l(e) \setminus \{\beta\}) \cup \{\beta_1\}$ and $l(e_2) = (l(e) \setminus \{\beta\}) \cup \{\beta_2\}$.
- (3) If a node e has one child e' , then there is an α -formula in $l(e)$ such that $l(e') = (l(e) \setminus \{\alpha\}) \cup \{\alpha_1, \alpha_2\}$ holds.

A leaf node is *closed* if its label contains complementary literals. A leaf is *incomplete* if its label contains nonliterals. A leaf is *open* if it is neither incomplete nor closed.

A tableau is *incomplete* if any of its leaf nodes is incomplete and *complete* otherwise; it is *closed* if all of its leafs are closed; and it is *open* if any of its leaf nodes are open.

A tableau is said to be *for* p , when its root is labelled with the singleton set $\{p\}$.

The definition of a tableau suggests an obvious algorithm for its construction. We will not explicate it here.

METAPROPOSITION 3.27 (Finiteness). *Every sentential semantic tableau is finite.*

PROOF. It is sufficient to note that the total number of subformulae in formulas in the label of each node is strictly smaller than the corresponding number in the parent node. \square

METACOROLLARY 3.28 (Computability). *The problem of constructing a complete sentential tableau for a sentential formula is computable.*

PROOF. Prove using finiteness and the obvious algorithm. \square

ALGORITHM 3.29 (Tableaux as a refutation procedure). Let the input be a sentential formula. Construct a closed or complete semantic tableau for its negation. Output 1 if the tableau is closed and 0 if it is open.

We will now embark in a proof that this algorithm is a decidable, sound and complete inference system for the sentential language.

METACOROLLARY 3.30 (Decidability of tableaux refutation). *Algorithm 3.29 terminates for all inputs.*

PROOF. Follows from Metacorollary 3.28. \square

METADefinition 3.31 (Hintikka set). A set H of sentential formulae is a (*sentential*) *Hintikka set* (also known as a *downward saturated set*) if the following hold:

- (1) For any literal ℓ , if $\ell \in H$ holds, then $\bar{\ell} \in H$ does not hold.

- (2) For any sentential formula p , if $\neg\neg p \in H$ holds, then $p \in H$ holds.
- (3) If $\alpha \in H$ holds, then both $\alpha_1 \in H$ and $\alpha_2 \in H$ hold.
- (4) If $\beta \in H$ holds, then at least one of $\beta_1 \in H$ and $\beta_2 \in H$ holds.

METADEFINITION 3.32. A set of sentential formulae is *satisfiable* if there is a truth valuation in which every formula in the set is true and *unsatisfiable* otherwise.

METALEMMA 3.33 (Hintikka's lemma). *Every Hintikka set is satisfiable.*

PROOF. Let H be the Hintikka set being considered and let L be the set of literals in H . Let ϕ be a truth valuation such that for each sentential variable x , $\phi(x) = 1$ holds if $x \in L$ holds, and $\phi(x) = 0$ holds if $\neg x \in L$ holds. Since a Hintikka set does not contain complementary literal, ϕ is well-defined, even if it is underspecified (choose any function that fits). The rest of the proof, left as an exercise, then shows that all members of H are true under ϕ . \square

METAPROPOSITION 3.34. *Let e_1, \dots, e_n be an open branch of a semantic tableau. Then $e_1 \cup \dots \cup e_n$ is a Hintikka set.*

PROOF. \square

METACOROLLARY 3.35 (Tableaux model property). *For every open leaf of a semantic tableaux for p there is a truth valuation that makes p true.*

PROOF. Combine the two previous lemmas. \square

METACOROLLARY 3.36 (Tableau completeness). *Algorithm 3.29 outputs 1 for all valid sentential formulae.*

PROOF. Assume that the algorithm outputs 0 for a valid formula p . Then, there is an open tableau for $\neg p$. By Metacorollary 3.35, then, $\neg p$ is satisfiable, which means that p is refutable and hence not valid. Thus we arrive at a contradiction. \square

METAPROPOSITION 3.37 (Tableau soundness). *If Algorithm 3.29 outputs 1 for a formula, it is valid.*

PROOF. It suffices to prove that if a sentential formula p has a closed tableau, it is unsatisfiable.

We will now prove by structural induction that in all closed tableaux, every node is labelled with an unsatisfiable set. From this the proposition follows trivially.

Let e be a leaf node. Since the tableau is closed, the node is closed and thus $l(e)$ contains complementary literals. Therefore, $l(e)$ is unsatisfiable.

Let e be a node with one child e' . We make the induction assumption that $l(e')$ is unsatisfiable. Now, by definition, $l(e) = (l(e) \setminus \{\alpha\}) \cup$

$\{\alpha_1, \alpha_2\}$ holds for some $\alpha \in l(e)$. Since $l(e')$ is unsatisfiable, either or both of $l(e) \setminus \{\alpha\}$ and $\{\alpha_1, \alpha_2\}$ are unsatisfiable. If the former is, then $l(e)$ is surely also unsatisfiable. If $\{\alpha_1, \alpha_2\}$ is unsatisfiable, then at least one of α_1 and α_2 is unsatisfiable. Since $\alpha \equiv \alpha_1 \wedge \alpha_2$ holds, α is then unsatisfiable and thus $l(e)$ is also unsatisfiable.

Let e be a node with two children, e_1 and e_2 . We make the induction assumption that $l(e_1)$ and $l(e_2)$ are unsatisfiable. Now, by definition, $l(e_i) = (l(e) \setminus \{\beta\}) \cup \{\beta_i\}$ holds for some $\beta \in l(e)$ and all $i = 1, 2$. Since $l(e_i)$ are unsatisfiable, either or both of $l(e) \setminus \{\beta\}$ and $\{\beta_i\}$ are unsatisfiable. If the former is, then $l(e)$ is surely also unsatisfiable. Otherwise, both of $\{\beta_i\}$ are unsatisfiable, and both of β_i are unsatisfiable. Since $\beta \equiv \beta_1 \vee \beta_2$ holds, β is then unsatisfiable and thus $l(e)$ is also unsatisfiable. \square

4.2. Davis–Putnam procedure. The Davis–Putnam procedure is one of the oldest refutation procedures for sentential languages and also one of the fastest. It operates on a set of sets of clauses, by transforming it according to four transformation rules. We will assume that no clause we discuss in this subsection contains complementary literals or the sentential constants \top and \perp . This assumption is easy to satisfy by simple rewrites of the clauses.

METADefinition 3.38 (One-literal rule). Let ℓ be a literal, let S be a set of clauses containing $\{\ell\}$ and let B be a set of sets of clauses containing S . Now the *one-literal rule*, given B , S and ℓ as input, outputs B so modified that from S the clause $\{\ell\}$ is removed and from all clauses of S the literal $\bar{\ell}$ is removed. The rule cannot be applied to any other kind of output.

METAPROPOSITION 3.39 (Soundness of one-literal rule). *If B' is the output of one-literal rule given B , S and ℓ as input, then B is satisfiable if and only if B' is satisfiable.*

PROOF. Omitted. \square

METADefinition 3.40 (Affirmative-negative rule). Let ℓ be a literal and let S be a set of clauses, some of which contain ℓ and none of which contain $\bar{\ell}$. Further, let B be a set of sets of clauses containing S . Now the *affirmative-negative rule*, given B , S and ℓ as input, outputs B so modified that from S all clauses containing ℓ are removed. The rule cannot be applied to any other kind of output.

METAPROPOSITION 3.41 (Soundness of affirmative-negative rule). *If B' is the output of affirmative-negative rule given B , S and ℓ as input, then B is satisfiable if and only if B' is satisfiable.*

PROOF. Omitted. \square

METADefinition 3.42 (Subsumption rule). Let S_1 and S_2 be two sets of clauses where $S_1 \subset S_2$ holds and let B be a set of sets of clauses

containing S_1 and S_2 . Now the *subsumption rule*, given B , S_1 and S_2 as input, outputs $B \setminus S_1$. The rule cannot be applied to any other kind of output.

METAPROPOSITION 3.43 (Soundness of subsumption rule). *If B' is the output of subsumption rule given B , S_1 and S_2 as input, then B is satisfiable if and only if B' is satisfiable.*

PROOF. Omitted. \square

METADefinition 3.44 (Splitting rule). Let ℓ be a literal and let S be a set of clauses, some of which contain ℓ and some of which contain $\bar{\ell}$. Further, let B be a set of sets of clauses containing S . We define:

$$S_\ell = \{ C \mid \ell \notin C \wedge \exists C' \in S : C = C' \setminus \{\bar{\ell}\} \}$$

$$S_{\bar{\ell}} = \{ C \mid \bar{\ell} \notin C \wedge \exists C' \in S : C = C' \setminus \{\ell\} \}$$

Now the *splitting rule*, given B , S and ℓ as input, outputs $(B \setminus S) \cup \{S_\ell, S_{\bar{\ell}}\}$. The rule cannot be applied to any other kind of output.

METAPROPOSITION 3.45 (Soundness of splitting rule). *If B' is the output of splitting rule given B , S and ℓ as input, then B is satisfiable if and only if B' is satisfiable.*

PROOF. Omitted. \square

METAPROPOSITION 3.46 (Davis–Putnam completeness). *Denote the operation that takes B as input and outputs what the one-literal rule outputs given B , S and ℓ as input as $O(S, \ell)$.*

Denote the operation that takes B as input and outputs what the affirmative-negative rule outputs given B , S and ℓ as input as $A(S, \ell)$.

Denote the operation that takes B as input and outputs what the subsumption rule outputs given B , S_1 and S_2 as input as $U(S_1, S_2)$.

Denote the operation that takes B as input and outputs what the splitting rule outputs given B , S and ℓ as input as $S(S, \ell)$.

A Davis–Putnam proof of a clause form C is a finite sequence of the signs $O(S, \ell)$, $A(S, \ell)$, $U(S_1, S_2)$ and $S(S, \ell)$, where S , ℓ and S_1 and S_2 are replaced by appropriate sets or literals, which, when interpreted as a pipeline where the n th operation takes as input $n - 1$ th operation's output, transforms $\{C\}$ into a set of sets of clauses where every clause is empty.

A sentential formula p is valid if and only if it has a Davis–Putnam proof.

PROOF. Omitted. \square

An algorithmic way of searching for a Davis–Putnam proof is given by always trying rules in the order they are presented here, backtracking as necessary.

4.3. Sentential resolution. The *sentential resolution rule* operates on a clause form of a sentential formula as follows:

Assume that the clause form contains two clauses C_1 and C_2 such that a literal ℓ is in C_1 and its complement $\bar{\ell}$ is in C_2 . Add to the clause form the clause $C_1 \cup C_2 \setminus \{\ell, \bar{\ell}\}$.

The *sentential resolution procedure* takes a sentential formula and applies the ground resolution rule to it repeatedly, stopping as soon as the clause form contains an empty clause, or until no application of the rule changes the clause form. If the resulting clause form contains an empty clause, the output is 1 and otherwise 0. This is a refutation procedure: when it outputs 1, the negation of the input is valid, and vice versa.

4.4. On heuristics. In all algorithms given above, making choices play an important part. They are *nondeterministic* algorithms. In many of them, the nondeterminism is *demonic*, meaning that they work regardless of how the choices are made (as long as each is a valid choice). However, their efficiency is critically affected by the method of choice. In a practical automated reasoning system, choices must be controlled by intelligently chosen heuristics. We will not discuss them here any further.