[**1 Truth Tables**](#_yrmgno93htcl) **2**

[1.1 Propositional Languages](#_exadxcb6g1jd) 2

[1.2 Truth Functions](#_vnv8i8q32mtm) 2

[1.3 Functional Completeness](#_aoj50mqxa9vd) 3

[**2 SAT Problems**](#_rf1m3qjgbed) **3**

[2.1 General Considerations](#_sr2l5kouo84s) 3

[2.2 Normal Forms](#_9bpnpmnn30yl) 3

[2.2.1 Negation Normal Forms](#_6bb1695lqf6v) 4

[2.2.2 Conjunctive Normal Forms](#_8o37m6ry1onr) 4

[2.3 Resolution Calculus](#_qjr8sdjlzcqa) 4

[2.4 DPLL Procedure](#_1rr8ln7v1apc) 5

[2.4.1 Examples](#_f5q8mk3t6e7) 5

[2.5 SAT-solvers](#_hsevfcb45gyi) 5

[2.5.1 Exercises and Applications](#_ylvz3081q0rd) 5

[2.6 Appendix: Cook-Levin Theorem](#_pcjklcpe6kts) 5

[**3 Search, Conflict and Backjumping**](#_2h844k85nxk8) **5**

[3.1 Search and Conflict](#_pt4ezolp9ux5) 5

[3.2 Search Rules](#_gcsenaqjibnk) 5

[3.3 Conflict Rules](#_lm6sroih69d) 6

[3.4 Examples](#_pm9jsnspn6vg) 6

[**4 Semantics for First-Order Languages**](#_kfcmylfjprrg) **6**

[4.1 Motivations](#_cg41jj4su8y9) 6

[4.2 Elementary Languages](#_uo3vj2nbpnn) 7

[4.3 Structures](#_8t5rpvcyj1bj) 8

[4.4 Ground Clauses Satisfiability](#_cibih97xxqno) 9

[4.5 Herbrand’s Theorem](#_ywlhbh7m0iyn) 9

[4.6 Complements and Examples](#_wse9n871ga8z) 9

[4.7 Skolemization](#_eyrbuoawu13v) 9

[**5. Satisfiability Modulo Theories**](#_259omgha97mx) **9**

[5.1 CDCL(T) procedure](#_s2i5sc7852aa) 9

[5.2 Examples of decision procedures](#_8qd1iian9gp0) 9

[5.3 Combination of decision procedures](#_484sntfhmjtt) 9

[6 Using an SMT-solver 78](#_q0s6v3rg3oi4) 9

[6.1 The SMT-LIB2 standard](#_t2iwgql32c0t) 9

[6.1.1 Complements and Examples](#_yy5wawh82qbz) 9

[6.2 Transition Systems and Bounded Model Checking](#_1p0d1ayyqhkc) 9

[6.2.1 Pre- and Post-Conditions](#_hjc1rv2dk7zm) 9

# 

# 1 Truth Tables

## 1.1 Propositional Languages

**Sentence:** any proposition for which it makes sense to ask whether it is true or false

**Atomic sentence:** A sentence if it contains no other sentence as its proper part

**Composite sentence:** If a sentence contains other sentences

**Connectives:** words like “and” “or” “if” … “then” “not” “but”. They allow to obtain composed sentences from simple sentences

**Formula:** expression of the type *p^q* or *p v q*… Are representations in an artificial language of the conceptual content of natural language sentences.

**Propositional language L:** A set whose elements are called propositional letters (p,q, r…) it is assumed to be finite or countable

**L-formulas:** We call L-formulas or formulas only the symbol strings that are obtained by applying finitely many times the following formation rules

**Atomic formula:** every p ∈ L is a formula (this is said to be an atomic formula)

**Uniqueness of reading:** reconstruct in a unique way the construction procedure of a formula

**Sub(A):** indicates the set of the subformulas of the formula A

## 1.2 Truth Functions

**Tautology (logical truth):** Whenever we have V(A) = 1 for every possible V. A tautology is a proposition that is always true (i.e. its truth-value is always T)

**Contradiction (refutable):**  Whenever we have V(A) = 0 for every V. A contradiction is a formula that is always false

**Satisfiable:** A is satisfiable iff we have V(A) = 1 for at least one V

## 1.3 Functional Completeness

**Functional completeness (theorem):** *Every truth function ψ having n ≥ 1 arguments is of the kind φA for some formula A(p1, . . . , pn)*

**DNF (Disjunctive normal forms):** disjunction of conjunctions, where every conjunct is an atomic formula or a negated atomic formula

# 2 SAT Problems

## 2.1 General Considerations

**SAT - Problem:** A SAT - Problem states, given a general propositional formula, if there exists an assignment that makes that formula true. At the moment, all SAT algorithms implemented in computers require exponential time to solve such problems.

**Saturation:** This technique is used by algorithms that try to complete all the available data until they find a contradiction; if there are no contradictions the satisfiability is ensured. This technique is used by the resolution algorithm.

**Optimize control of backtracking:** This technique is used by algorithms based on the association of all given propositional letters with a True or False value. When the association produces a contradiction the algorithm backtracks. An example of this algorithm is the DPLL.

## 2.2 Normal Forms

**Atoms = Atomic formulas.**

**Literals:**  Atomic formulas and their negations.

**Clauses:** Disjunction of literals.

**☐** **:** Empty clause.

**Logical equivalence:** Two formulas A and B are logically equivalent iff A ⇔ B is a tautology, so iff V(A) = V(B) for every possible V. They have the same truth table.

**Negation Normal Form:** A formula is in negation normal form iff does not contain implications and iff the negation appears only in front of atomic formulas.

**Conjunctive normal form:** A formula is in conjunctive normal form iff it’s a conjunction of clauses.

## 2.2.1 Negation Normal Forms

**Rules to transform any formula in NNF:**

X → Y = ¬ X v Y

¬ (X ^ Y) = ¬ X v ¬ Y

¬ (X v Y) = ¬ X ^ ¬ Y

¬ ¬ X = X

## 2.2.2 Conjunctive Normal Forms

**Methods:** The non-structural method transforms a formula in cnf requiring exponential time. The structural method transforms a formula in a cnf formula requiring the same order of magnitude of the first one.

**Rules for non structural method:**

X ^ ( Y v Z) = (X ^ Y) v (X ^ Y)

X v (Y ^ Z) = (X v Y) ^ (X v Y)

## 2.3 Resolution Calculus

**Resolution Algorithm:**

* Takes a CNF and NNF set of clauses C\_0 = {C1,....,Ck} and separates all negatives literals from positive ones with an arrow => (negatives at left and positives at right):

p1,....,pn => q1,.....qn

* applies the **resolution rule:** Takes a random couple of clauses written in the above notation, if a literal appears at a side of the arrow in a clause and in the opposite side in the other one gets cancelled;
* The Algorithm applies this rule in all possible ways, it stops only if the **Empty Clause** appears or if there are no more possibilities, so if the set of clauses is **Saturated.**

**Refutational Completeness Theorem:** Let C0 be a set of clauses; we have that C0 is

inconsistent iff the empty clause belongs to R(C0). (where R(C0) is C0 after the application of the Resolution Algorithm).

**Subsumption rule:** If the resolution rule is applied to two clauses like P=>Q and P’=>Q’ and P subsumed P’, Q subsumed Q’, the P’=>Q’ can be eliminated without compromising the consistency of the set of clauses.

**Horn Clause:** If the set of clauses is composed only of horn clauses (only one positive literal) the computational complexity becomes linear instead of exponential.

## 2.4 DPLL Procedure

**DPLL Procedure:** executes all deterministic operations on a set of clauses useful to propagate known information. When there are no more possible operations executable, assign a truth value to a propositional letter and repeat the procedure.

## 2.4.1 Examples

## 2.5 SAT-solvers

## 2.5.1 Exercises and Applications

## 2.6 Appendix: Cook-Levin Theorem

# 3 Search, Conflict and Backjumping

## 3.1 Search and Conflict

**(V | F | α):**

* **α:** ia a clause (to be called a conflict clause) or it is the conventional byte ∗; if α = ∗ we say that the system is in search status, otherwise we say that the system is in conflict state.
* **F**: is a finite set of clauses.
* **V:** is a partial assignment, whose literals are stored as an ordered list and are all marked by either the label ‘decided’ or the label ‘propagated’; we write l(d) to say that l is decided (if l is propagated, we write l(p) or just l).

**Unsatisfiability**: If the algorithm reaches a status (V | F | C) where it is possible to apply the failing rule, then F (and consequently also F0) is not satisfiable.

**Satisfiability**: If the algorithm reaches a status of the kind (V | F | ∗) where no rule applies, then the clauses in F (and consequently also F0) are all satisfied by V

## 3.2 Search Rules

**Propagate rule**: if we are in the status of searching (\*), we can propagate a new literal l(p) in (l(0), l(p) | F | \*) if in our clauses(V1, V2, V3…) is present a clause (V0) that contains the negation of l(0). This allows us to define absolutely the propagation of l by the clause V0 . If we are at the beginning and it is present clause (V0) formed by a unique literal (l), then l is propagated by the clause V0.

**Decide Rule:** if we are in the status of searching (\*), we can propagate a new literal l(d) if:

* at the beginning in ( | F | \*), our clauses (V1, V2, V3…) does not contain any clause (V0) formed by a unique literal and so we have only clauses in which we have a choice (like V1= l v m, V2 = p v s)
* in (l(p) | F | \*), our clauses (V1, V2, V3…) does not contain the negation of the propagated literal (¬l(p)) and so we have to decide a new literal between our clauses

## 3.3 Conflict Rules

**Explain rule:** When we are in a conflict state (\*), like (l(1), l(2) | F | ¬l v m) the Explain rule is applied when one of the literals in the clause in the conflict state (¬l v m) is in conflict with the last propagated literal l(p). The Explain rule is applied with a chronological order, so when we have a conflict and we applied the resolution with the clause of the literal propagated (in that case V2) and the clause of the literal in conflict (¬l v m), the propagated literal l(2) is eliminated and the new clause (Vnew) formed by the resolution remains in the conflict state. If we apply the Explain rule with only propagated literals (so without decided literals) the process gets the empty clause so it is unsat.

**Backjumping rule:** When we are in a conflict state (\*) , the Backjumping rule is applied when the last literal in the clause in conflict is a decided literal. Compared to the Explain rule, the Backjumping rule could be applied without a chronological order: the last literal in the conflict, could not be the last decided literal. The literal in the conflict clause substitutes the decided literal and the conflict clause becomes a new learned clause. The literal is propagated by the new learned clause.

## 3.4 Examples

# 4 Semantics for First-Order Languages

In the first-order language we shall have the following symbols: t*he propositional connectives and the two quantifiers, individual variables x0, x1…, individual constants a1, a2, …, predicate letters P, Q . . ., function letters f, g …* We will consider individual constants as 0-ary functions and propositional variables as 0-ary predicates

## 4.1 Motivations

**Quantifiers:** ∀ and ∃ are two operators

**∀:** universal quantifier, *for every*. The expression ∀x P(x) means that “for every x the property property P holds”

**∃:** existential quantifier, *exist*. The expression ∃x P(x) meaning “there is an x enjoying the property P”

**Terms:** represents elements of the discourse domain (numbers, people, objects, etcs)

**Predicates:** represent relationships that can exist between these objects (being a man, being less than…). Allows us to express properties and relations on objects. P(x) indicates that x enjoys a given property P

## 4.2 Elementary Languages

Elementary language is richer than a propositional language and allows to name individuals, to construct designators of individuals from other designators of individuals

**Elementary language:** *L = ( P, F α )* is a triple comprising the following data:

* a set of predicate symbol P
* a set of function symbol F
* a function α P ∪ F → N called arity function

It is assumed that P and F are disjoint sets

**Arity function α:** associates to each predicate or function symbol the number of its arguments

**Individual variables:** set V = {x0, x1, x2, … , y0, y1, y2, … }

**Connectives:** ∧, ∨,→, ¬

**Quantifiers:** ∀ ("for every") and ∃ ("exists")

**Set of L-terms**: (or simply terms)

* every x ∈ V and every c ∈ F0 is a term
* if f ∈ Fn (for n ≥ 1) and t1, … , tn are terms, f(t1, … , tn) is a term

**Set of L-formulas:** (or simply formulas)

* (P(t1, … , tn)) is an (atomic) formula
* if A1, A2 are formulas, so are (A1 ∧ A2), (A1 ∨ A2), (A1 → A2), (¬A1)
* if A is a formula and x ∈ V, then (∀xA) and (∃xA) are formulas

**Ground:** a term is closed (or ground) if it is built up without using variables. Similarly, a formula is ground iff it does not contain variables.

**Set of subterms of a term t**: Sub(t) = {t}, if t ∈ V ∪F0, Sub(f(t1, … , tn)) = {ft1, … , tn)} ∪ Sub(t1) ∪…∪ Sub(tn)

**Bounded:** An occurrence of a variable x in a formula A is said to be bounded if it is located inside a subformula of the form ∀xB or ∃xB. If this is not the case, the occurrence is said to be **free**. A variable x is said to be free in A if some occurrence of it in A is free. With the notation A(x1, … xn) we say that the variables that occur free in A are between x1, … xn

**x, y, … :** Indicate *n*-tuples of variables of unspecified length and **t, u, …** for n-tuples of terms. n-tuples of variables like x, y, … are always implicitly assumed not to contain repetitions. t, u, … that therefore indicate n-tuples of terms with possible repetitions

**Sentence (or closed formula):** is a formula in which no variable is free. If A has exactly the variables x1, … , xn as free variables, then A∀ (the universal closure of A) denotes the sentence ∀x1 · · · ∀xnA. Similarly, A∃ (the existential closure of A) denotes the sentence ∃x1…∃xnA.

## 4.3 Structures

PDF PAGE 53

## 4.4 Ground Clauses Satisfiability

## 4.5 Herbrand’s Theorem

## 4.6 Complements and Examples

## 4.7 Skolemization

# 5. Satisfiability Modulo Theories

## 5.1 CDCL(T) procedure

## 5.2 Examples of decision procedures

## 5.3 Combination of decision procedures

## 6 Using an SMT-solver 78

## 6.1 The SMT-LIB2 standard

## 6.1.1 Complements and Examples

## 6.2 Transition Systems and Bounded Model Checking

## 6.2.1 Pre- and Post-Conditions