Foundations of Artificial Intelligence

31. Propositional Logic: DPLL Algorithm

Malte Helmert and Gabriele Röger

University of Basel

April 24, 2017

Foundations of Artificial Intelligence

April 24, 2017 — 31. Propositional Logic: DPLL Algorithm

Propositional Logic: Overview

Chapter overview: propositional logic

- 29. Basics
- 30. Reasoning and Resolution
- 31. DPLL Algorithm
- 32. Local Search and Outlook

31.1 Motivation

Propositional Logic: Motivation

- Propositional logic allows for the representation of knowledge and for deriving conclusions based on this knowledge.
- many practical applications can be directly encoded, e.g.
  - constraint satisfaction problems of all kinds
  - circuit design and verification
- many problems contain logic as ingredient, e.g.
  - automated planning
  - general game playing
  - description logic queries (semantic web)

Propositional Logic: Algorithmic Problems

main problems:

- reasoning ($\Theta \models \varphi$?):
  Does the formula φ logically follow from the formulas Θ?
- equivalence ($\varphi \equiv \psi$):
  Are the formulas φ and ψ logically equivalent?
- satisfiability (SAT):
  Is formula φ satisfiable? If yes, find a model.

German: Schlussfolgern, Äquivalenz, Erfüllbarkeit

The Satisfiability Problem

The Satisfiability Problem (SAT)

given:
propositional formula in conjunctive normal form (CNF)
usually represented as pair $\langle V, \Delta \rangle$:

- V set of propositional variables (propositions)
- Δ set of clauses over V
  (clause = set of literals v or $\neg v$ with $v \in V$)

find:

- satisfying interpretation (model)
- or proof that no model exists

SAT is a famous NP-complete problem (Cook 1971; Levin 1973).

Relevance of SAT

- The name "SAT" is often used for the satisfiability problem for general propositional formulas (instead of restriction to CNF).
- General SAT can be reduced to CNF
  (conversion in time $O(n)$).
- All previously mentioned problems can be reduced to SAT
  (conversion in time $O(n)$).

$\rightsquigarrow$ SAT algorithms important and intensively studied

this and next chapter: SAT algorithms

31.2 Systematic Search: DPLL

SAT vs. CSP

SAT can be considered as constraint satisfaction problem:

- CSP variables = propositions
- domains = $\{\mathbf{F}, \mathbf{T}\}$
- constraints = clauses

However, we often have constraints that affect > 2 variables.

Due to this relationship, all ideas for CSPs are applicable to SAT:

- search
- inference
- variable and value orders

The DPLL Algorithm

The DPLL algorithm (Davis/Putnam/Logemann/Loveland) corresponds to backtracking with inference for CSPs.

- recursive call DPLL(Δ, I) for clause set Δ and partial interpretation I
- result is consistent extension of I; **unsatisfiable** if no such extension exists
- first call DPLL($\Delta, \emptyset$)

inference in DPLL:

- simplify: after assigning value d to variable v, simplify all clauses that contain v
  $\rightsquigarrow$ forward checking (for constraints of potentially higher arity)
- unit propagation: variables that occur in clauses without other variables (unit clauses) are assigned immediately
  $\rightsquigarrow$ minimum remaining values variable order

The DPLL Algorithm: Pseudo-Code

function DPLL(Δ, I):

if $\square \in \Delta$: [empty clause exists $\rightsquigarrow$ unsatisfiable]
    return unsatisfiable
else if $\Delta = \emptyset$: [no clauses left $\rightsquigarrow$ interpretation I satisfies formula]
    return I
else if there exists a unit clause $\{v\}$ or $\{\neg v\}$ in Δ: [unit propagation]
    Let v be such a variable, d the truth value that satisfies the clause.
    $\Delta' :=$ simplify(Δ, v, d)
    return DPLL($\Delta', I \cup \{v \mapsto d\}$)
else: [splitting rule]
    Select some variable v which occurs in Δ.
    for each $d \in \{\mathbf{F}, \mathbf{T}\}$ in some order:
        $\Delta' :=$ simplify(Δ, v, d)
        $I' :=$ DPLL($\Delta', I \cup \{v \mapsto d\}$)
        if $I' \neq$ **unsatisfiable**
            return I'
    return unsatisfiable

The DPLL Algorithm: simplify

function simplify(Δ, v, d)

Let ℓ be the literal for v that is satisfied by $v \mapsto d$.
Let $\bar{\ell}$ be the complementary (opposite) literal to ℓ.
$\Delta' := \{C \mid C \in \Delta \text{ such that } \ell \notin C\}$
$\Delta'' := \{C \setminus \{\bar{\ell}\} \mid C \in \Delta'\}$
return Δ''

Example (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. unit propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. splitting rule:

2a. $X \mapsto \mathbf{F}$
$\{\{Y\}, \{\neg Y\}\}$

3a. unit propagation: $Y \mapsto \mathbf{T}$
$\{\square\}$

2b. $X \mapsto \mathbf{T}$
$\{\{\neg Y\}\}$

3b. unit propagation: $Y \mapsto \mathbf{F}$
$\{\}$

Example (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. unit propagation: $Z \mapsto \mathbf{T}$
   $\{\{W, \neg X, \neg Y\}, \{X\}, \{Y\}\}$
2. unit propagation: $X \mapsto \mathbf{T}$
   $\{\{W, \neg Y\}, \{Y\}\}$
3. unit propagation: $Y \mapsto \mathbf{T}$
   $\{\{W\}\}$
4. unit propagation: $W \mapsto \mathbf{T}$
   $\{\}$

Properties of DPLL

- DPLL is sound and complete.
- DPLL computes a model if a model exists.
  - Some variables possibly remain unassigned in the solution I; their values can be chosen arbitrarily.
- time complexity in general exponential

$\rightsquigarrow$ important in practice: good variable order and additional inference methods (in particular clause learning)

- Best known SAT algorithms are based on DPLL.

31.3 DPLL on Horn Formulas

Horn Formulas

important special case: Horn formulas

Definition (Horn formula)

A Horn clause is a clause with at most one positive literal, i.e., of the form

$$\neg x_1 \vee \dots \vee \neg x_n \vee y \text{ or } \neg x_1 \vee \dots \vee \neg x_n$$

($n = 0$ is allowed.)

A Horn formula is a propositional formula in conjunctive normal form that only consists of Horn clauses.

German: Hornformel

- foundation of logic programming (e.g., PROLOG)
- hot research topic in program verification

DPLL on Horn Formulas

Proposition (DPLL on Horn formulas)

If the input formula φ is a Horn formula, then the time complexity of DPLL is polynomial in the length of φ.

Proof.

properties:

1. If Δ is a Horn formula, then so is simplify(Δ, v, d). (Why?)
   $\rightsquigarrow$ all formulas encountered during DPLL search are Horn formulas if input is Horn formula
2. Every Horn formula without empty or unit clauses is satisfiable:
   - all such clauses consist of at least two literals
   - Horn property: at least one of them is negative
   - assigning **F** to all variables satisfies formula

...

DPLL on Horn Formulas (Continued)

Proof (continued).

3. From 2. we can conclude:
   - if splitting rule applied, then current formula satisfiable, and
   - if a wrong decision is taken, then this will be recognized without applying further splitting rules (i.e., only by applying unit propagation and by deriving the empty clause).
4. Hence the generated search tree for n variables can only contain at most n nodes where the splitting rule is applied (i.e., where the tree branches).
5. It follows that the search tree is of polynomial size, and hence the runtime is polynomial.

□

31.4 Summary

Summary

- satisfiability basic problem in propositional logic to which other problems can be reduced
- here: satisfiability for CNF formulas
- Davis-Putnam-Logemann-Loveland procedure (DPLL): systematic backtracking search with unit propagation as inference method
- DPLL successful in practice, in particular when combined with other ideas such as clause learning
- polynomial on Horn formulas (= at most one positive literal per clause)