Properties of SLUR Formulae

Ondřej Čepek, Petr Kučera, Václav Vlček
Charles University in Prague

SOFSEM 2012
January 23, 2012

Outline

- Boolean terminology (quick recap)
- Definition of the SAT problem
- General case of SAT
  - NP-completeness (Cook-Levin Theorem)
  - SAT solvers (and how they work)
- Special cases of SAT
  - Quadratic CNFs
  - Horn CNFs
  - Hidden Horn CNFs
  - SLUR CNFs

Boolean basics

Literals, clauses, CNFs, implicates

- Boolean function on n variables is a mapping $\{0,1\}^n \rightarrow \{0,1\}$
- Literal = variable or its negation
- Clause = disjunction of literals (no complementary pair)
- Conjunctive Normal Form (CNF) = conjunction of clauses (Fact: Every Boolean function has a CNF representation)
- Clause C is an implicate of CNF F if F = 1 implies C = 1. C is a prime implicate of F if removing an arbitrary literal from C produces a clause which is not an implicate of F.
- A CNF is prime if it consists only of prime implicates.
- Canonical CNF = CNF consisting of all prime implicates (= constant for a tautology and for an unsatisfiable CNF)

Boolean basics

Resolution and its completeness

- Two clauses are resolvable $\Leftrightarrow$ they have exactly one conflicting literal, e.g. $C_1 = A \vee x$ and $C_2 = B \vee \neg x$ where A and B have no conflicting literal
- Product of a resolution (of two resolvable caluses) = resolvent, e.g. $R(C_1, C_2) = A \vee B$
- A resolvent of two implicates of f is also an implicate of f
- Notation: $\mathbf{R}(S)$ = a resolution closure of set S of clauses (it may take an exponential time to generate $\mathbf{R}(S)$)
- Resolution is complete [Quine's theorem (1955)]: Let f be a Boolean function, S a set of clauses in an arbitrary CNF representation of f, and P the set of all prime implicates (canonical CNF) of f. Then $P \subseteq \mathbf{R}(S)$.

Boolean basics

Unit Propagation

- Unit resolution = at least one of the parent clauses is a unit clause (contains exactly one literal)
- Unit propagation = iterative use of unit resolution until no unit clause exists or contradiction (empty clause) is derived or a CNF where every clause is at least quadratic.
- Implementation: repeatedly fix a variable value forced by a unit clause, drop all clauses containing the assigned literal, and drop all occurrences of the complementary literal (which shortens the corresponding clause).
- Complexity: O(l) where l is the number of literals in the input CNF.
- Unlike general resolution unit propagation is not complete

SAT problem

Definition

Input: A CNF F on n Boolean variables $x_1, \ldots, x_n$.

Question: Does there exist a truth assignment to $x_1, \ldots, x_n$ which satisfies F?

- SAT is one of the most basic and most studied problems in computer science
- It has many practical applications in VLSI design, network design (and in many other fields where Boolean variables naturally describe the studied problem)
- A procedure which generates resolution closure is enough to solve SAT (but it is exponential).
- How hard it is to solve SAT, i.e. what is the complexity of this problem?

SAT problem
General case

Theorem (Cook-Levin 1971): SAT is NP-complete.

How to solve SAT in practice? Heuristics!

There are both commercial and open source SAT solvers that employ very sophisticated heuristics to direct an exhaustive search over all truth assignments (variants of the DPLL procedure) and perform quite well even on large inputs of the SAT problem from practice.

Examples: SATZ, GSAT, GRASP, CHAFF, ...

SAT problem

Solvable cases

There are many classes of CNFs for which polynomial time SAT algorithms are known:

- Quadratic CNFs
- Horn CNFs
- Hidden Horn CNFs
- Extended Horn
- Q-Horn CNFs
- Matched CNFs
- SLUR CNFs

… and many more

SAT algorithm (non-deterministic)
Single Lookahead Unit Resolution

SLUR(F) {F is CNF with no empty clause}
$(F,t) \leftarrow$ UnitProp(F) {t is a partial truth assignment}
if F contains an empty clause (EC) return UNSAT and stop
while F is not empty do {loop while F contains clauses}
select at random a literal v appearing in F
$(F_1,t_1) \leftarrow$ UnitProp($F \wedge \neg v$)
$(F_2,t_2) \leftarrow$ UnitProp($F \wedge v$)
if both F_1 and F_2 contain EC GIVE UP and stop
if exactly one of F_1 and F_2 (say F_k) contains EC then
set $j \leftarrow 3\text{-}k$ and $(F,t) \leftarrow (F_j, t \cup t_j)$
if none of F_1 and F_2 contains EC then select j at
random and set $(F,t) \leftarrow (F_j, t \cup t_j)$
return SAT and the satisfying assignment t

SLUR class of CNFs

Properties

Definition: (Franco, Van Gelder 2003) A CNF F belongs to the SLUR class if the SLUR algorithm never gives up on F regardless of the sequence of random choices of literals.

Key fact: A CNF F is SLUR $\Leftrightarrow$ whenever a partial substitution into F produces an unsatisfiable CNF F' then unit propagation on F' derives an empty clause.

Fact 1: Horn CNFs $\subseteq$ Hidden Horn CNFs $\subseteq$ SLUR CNFs

Fact 2: Extended Horn CNFs $\subseteq$ SLUR CNFs

Fact 3: CC-balanced CNFs $\subseteq$ SLUR CNFs

Fact 4: Propagation complete CNFs $\subseteq$ SLUR CNFs

Horn CNFs and Horn functions

Properties

Definition: A CNF is Horn if every clause in it contains at most one positive literal (non-negated variable).

Fact 1: The Horn property of a CNF is preserved under partial substitution of truth values to variables.

Fact 2: Every Horn CNF not containing a unit clause is satisfiable (by an all-zero vector) and hence every unsatisfiable Horn CNF is detected by unit propagation. Thus the SLUR algorithm never gives up on a Horn CNF.

Definition: A Boolean function is Horn if it can be represented by a Horn CNF.

Fact: All prime implicates of a Horn function are Horn clauses. Thus any prime CNF of a Horn function is Horn.

SLUR class of CNFs

Questions

Question 1: Characterization of SLUR CNFs (syntactic, structural, any other than by an algorithm)?

Question 2: Recognition of SLUR CNFs (given a CNF can we effectively decide whether it is SLUR or not)?

Question 3: Does it make sense to make an analogy to Horn (hidden Horn, …) function definitions (A Boolean function is SLUR if it can be represented by a SLUR CNF)?

We will give answers to Question 3 and Question 2 (which will also partly answer Question 1)

SLUR class of CNFs

Answers

Proposition 1: Let F be a CNF. Then it is co-NP complete to recognize whether F is SLUR or not.

Proof: reduction from 3D matching (technical and long)

Proposition 2: Let f be a Boolean function and F be a canonical CNF representing f. Then F is a SLUR CNF.

Proof: based on a technical lemma which says that if we substitute into a canonical CNF then the result contains the canonical CNF of the restricted function

Corollary: SLUR functions = all Boolean functions

SLUR hierarchy

Definitions and properties

SLUR algorithm modification:

- pick j variables at random (instead of one)
- try out all 2^j assignments (if all of them fail then give up, otherwise select randomly one of the non-failing ones)

Modified algorithm defines a class SLUR(j) of CNFs

Properties: The recognition problem is co-NP complete for every class SLUR(j) and the hierarchy is non-collapsing:

$$\forall j : \mathrm{SLUR}(j) \subset \mathrm{SLUR}(j+1)$$

SLUR class

Miscellaneous notes

How “far” from a canonical form must a CNF be to lose the SLUR property?

Proposition: Let F be a CNF of f such that every prime implicate of f can be derived from F by at most one resolution step. Then F is in the SLUR class.

Fact: There exists a CNF F of f such that every prime implicate of f can be derived from F by at most two resolution steps and such that F is not in the SLUR class.

SLUR class

Miscellaneous notes

A CNF F is propagation complete if after any assignment of truth values into F we get for the obtained CNF G

d is an implicate of G $\Leftrightarrow$ d is derived from G by UnitProp for every unit clause d.

Proposition: PC CNFs $\subset$ SLUR CNFs

There are short Horn CNFs (4 variables, 2 clauses) which are not PC, so the containment is strict.

Thank you for your attention.