SAT and SMT
An Introduction

Jonathan Cederberg <jonathan.cederberg@it.uu.se>

Friday, April 20th, 2012
Outline

1. SAT
   - Complexity
   - DPLL

2. SMT
   - Current state of SAT and SMT
   - A Complete Example
Outline

1. SAT
   - Complexity
   - DPLL

2. SMT
“Given $f(p_1, \ldots, p_n)$ in propositional logic, is there an assignment to the propositional variables $p_1, \ldots, p_n$ such that $f$ evaluates to true?”

Decision problem

SAT is NP-complete (Cook, 1971)
SAT

- CNF: A conjunction of disjunctions of literals \((p_i \text{ or } \neg p_i)\):
  \[
  (l^1_1 \lor l^2_1 \lor \ldots \lor l^{k_1}_1) \land \\
  (l^1_2 \lor l^2_2 \lor \ldots \lor l^{k_2}_2) \land \\
  \vdots \\
  (l^1_m \lor l^2_m \lor \ldots \lor l^{k_m}_m)
  \]

- \(k\)-SAT, 3-SAT are NP-complete
- HORNSAT, 2-SAT are P-complete
SAT is NP-complete

- Nondeterministic TM finds a satisfying assignment in polynomial time.
- There is no (known) polynomial algorithm to find a satisfying assignment, only exponential.
- Checking if a given assignment is a satisfying one is polynomial.
- So, we need to search among the exponentially many assignments for a solution.
SAT-Solving

- Stålmarcks method (1994, US pat.nr. 5,276,897)
- Davis-Putnam-Logemann-Loveland (DPLL)-algorithm
DPLL, roughly

DPLL(f)
1   if $f$ is a consistent set of literals
2       then return True
3   if $f$ contains an empty clause
4       then return False
5   while there is a unit clause $l$ in $f$
6       do $f \leftarrow \text{Unit-Propagate}(l, f)$
7       $l \leftarrow \text{Choose-Literal}(f)$
8   return DPLL($f \land l$) or DPLL($f \land \neg l$)
Outline

1. SAT

2. SMT
   - Current state of SAT and SMT
   - A Complete Example
SMT

- SMT = Satifiability Modulo Theory
- A framework for solving satisfiability of formulas of some theory, where the theory is a parameter.
- “Layered” approach, based on SAT-solving.
SMT: example

\[ \neg(a \geq 3) \land (a \geq 3 \lor a \geq 5) \]
SMT: example

\[
\neg (a \geq 3) \land (a \geq 3 \lor a \geq 5)
\]

\[
\neg p \land (p \lor q)
\]
SMT: example

\[ \neg (a \geq 3) \land (a \geq 3 \lor a \geq 5) \]

\[ \neg p \land (p \lor q) \]

After abstraction, use SAT-solving to find candidate literals to satisfy using domain specific solver
SMT: example

¬(a ≥ 3) ∧ (a ≥ 3 ∨ a ≥ 5)

Initial suggestion from DPLL: ¬p and q should be true
SMT: example

- \( \neg(a \geq 3) \land (a \geq 3 \lor a \geq 5) \)

- Initial suggestion from DPLL:
  - \( \neg p \) and \( q \) should be true

- So \( \neg a \geq 3 \land a \geq 5 \) should be true
SMT: example

\[-(a \geq 3) \land (a \geq 3 \lor a \geq 5)\]

- Initial suggestion from DPLL: \(-p\) and \(q\) should be true
- Answer from domain specific solver: There is no model for \(-a \geq 3 \land a \geq 5\)
SMT: example

\[
\neg(a \geq 3) \land (a \geq 3 \lor a \geq 5)
\]

- Initial suggestion from DPLL: \(\neg p\) and \(q\) should be true
- Answer from domain specific solver: There is no model for \(\neg a \geq 3 \land a \geq 5\)
- From this we obtain “Theory Lemma”: \(a \geq 3 \lor \neg a \geq 5\), or \(p \lor \neg q\)
SMT: example

- \( \neg(a \geq 3) \land (a \geq 3 \lor a \geq 5) \)

- Initial suggestion from DPLL: \( \neg p \) and \( q \) should be true

- Answer from domain specific solver: There is no model for \( \neg a \geq 3 \land a \geq 5 \)

- From this we obtain “Theory Lemma”: \( a \geq 3 \lor \neg a \geq 5 \), or \( p \lor \neg q \)

- Add theory lemma as an additional clause to the original formula, and start over
SMT: example

- \( \neg p \land (p \lor q) \land (p \lor \neg q) \)

- Now DPLL responds “Unsatisfiable”.

\[\neg p \land (p \lor q) \land (p \lor \neg q)\]
SMT

A lot of different theories supported by various solvers

- Uninterpreted functions and constants
- Arithmetic (including nonlinear)
- Bit Vectors
- Arrays
- Datatypes (including recursive datatypes)
- ...
Current state of SAT and SMT

- They scale very well, (for SAT: hundreds of thousands of variables)
- They are used in a lot of applications:
  - Static Analys
  - Verification
  - Testing
A complete example: Test generation for path coverage

```python
def compare_stuff(a,b):
    if a < b:
        foo = "a is smaller"
    if b < a:
        foo = "b is smaller"
    print foo

if __name__ == '__main__':
    compare_stuff(1,2)
```
A complete example: Test generation for path coverage

Start by unfolding the program and collect the set of constraints for each path:

- $a > b \land b > a$
- $\neg(a > b) \land b > a$
- $a > b \land \neg(b > a)$
- $\neg(a > b \land \neg(b > a))$
A complete example: Test generation for path coverage

- Ask SMT solver for possible inputs:
  - \( a > b \land b > a \) Unsat, so no input can generate this path
  - \( \neg(a > b) \land b > a \) Sat, path generated by \( a = 0, b = -1 \)
  - \( a > b \land \neg(b > a) \) Sat, path generated by \( a = -1, b = 0 \)
  - \( \neg(a > b \land \neg(b > a)) \) Sat, path generated by \( a = 0, b = 0 \)
Exercises

1. Suggest a satisfiable propositional formula with 4 variables and an order of choosing literals, such that you need to backtrack at least once to find a satisfying assignment to the atoms. Show each step of the computation.

2. Use Z3 to prove that in propositional logic “proof by cases” is a valid way to prove things. Proof by cases can be expressed as “if $c$ is true whenever either $a$ or $b$ is true, then it must be the case that both $a$ implies $c$ and $b$ implies $c$.”
Questions?