The Barcelogic Research Group: Research Interests

Enric Rodríguez-Carbonell

COST Office

October 30, 2009

Departament de Llenguatges i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Overview of the talk

- Who are we?

Overview of the talk

- Who are we?
- Introduction to SAT and SMT

Overview of the talk

- Who are we?
- Introduction to SAT and SMT
- Research interests

Who are we?

- Research group of Universitat Politècnica de Catalunya (at Barcelona, Spain)
- Some of its members:

Robert Nieuwenhuis

Albert Oliveras

Albert Rubio

Javier Larrosa

Enric Rodríguez-Carbonell

...

Introduction to SAT and SMT . 1

- Historically, automated reasoning ≡ uniform proof-search procedures for FO logic
- Not a big success: is FO logic the best compromise between expressivity and efficiency?
- Current trend is to gain efficiency by:
  - addressing only (expressive enough) decidable fragments of a certain logic
  - incorporate domain-specific reasoning, e.g:
    - arithmetic reasoning
    - equality
    - data structures (arrays, lists, stacks, ...)

Introduction to SAT and SMT . 2

Examples of this recent trend:

- SAT: use propositional logic as the formalization language
  - + high degree of efficiency
  - - expressive (all NP-complete) but not natural encodings
- SMT: propositional logic + domain-specific reasoning
  - + improves the expressivity
  - - specific techniques need to be designed for each domain

Introduction to SAT

Problem definition:

- INPUT: propositional formula F
- OUTPUT: is F SATisfiable?

Example:

- $(p \vee q) \wedge (\overline{p} \vee q) \wedge (r \vee \overline{q})$ is SAT with model $\{q, p, r\}$
- $(p \vee q) \wedge (\overline{p} \vee q) \wedge (r \vee \overline{q}) \wedge (\overline{r} \vee \overline{q})$ is UNSAT

Simple but MANY applications:

- System verification
- Planning
- Scheduling
- ...

State of the art in SAT

- Main procedure: Davis-Putnam-Logemann-Loveland (DPLL) is depth-first search with backtracking
- Original procedure [DP'60, DLL'62] extended in the late 90's with:
  - conceptual improvements: backjumping, learning, ...
  - implementation techniques: 2-watched literals, cache-aware data structures, ...
- SAT solvers current capabilities: industrial instances with thousands of variables and millions of clauses

Introduction to SMT

- Some problems are more naturally expressed in other logics than propositional logic, e.g, in software verification
- SMT consists of deciding the satisfiability of a (ground) FO formula with respect to a background theory:
  - Equality with Uninterpreted Functions (EUF):
    $$g(a) = c \quad \wedge \quad (f(g(a)) \neq f(c) \ \vee \ g(a) = d) \quad \wedge \quad c \neq d$$
  - (Integer/Real) Difference Logic:
    $$(x - y \leq 1 \ \vee \ y - z \leq 0) \quad \wedge \quad x - z < 0$$
  - Linear (Integer/Real) Arithmetic:
    $$(x + y \leq 1 \ \wedge \ y - 2z \geq 0) \quad \vee \quad x + y + z > 4$$
  - Arrays: $A = write(B, a, 4) \ \wedge \ (read(A, b) = 2 \ \vee \ A \neq B)$
  - ...
  - Combinations:
    $A = write(B, a+1, 4) \ \wedge \ (read(A, b+3) = 2 \ \vee \ f(a-1) \neq f(b+1))$

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$
- Theory solver says T-inconsistent

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$
- Theory solver says T-inconsistent
- Send $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4\}$ to SAT solver

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$
- Theory solver says T-inconsistent
- Send $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $[1, 2, 3, \overline{4}]$

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$
- Theory solver says T-inconsistent
- Send $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $[1, 2, 3, \overline{4}]$
- Theory solver says T-inconsistent

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$
- Theory solver says T-inconsistent
- Send $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $[1, 2, 3, \overline{4}]$
- Theory solver says T-inconsistent
- SAT solver detects $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2, \overline{1} \vee \overline{2} \vee \overline{3} \vee 4\}$ UNSAT!

State of the art in SMT: lazy approach . 1

Example: consider EUF and

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad (\underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- SAT solver returns model $[1, \overline{2}, \overline{4}]$
- Theory solver says T-inconsistent
- Send $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4\}$ to SAT solver
- SAT solver returns model $[1, 2, 3, \overline{4}]$
- Theory solver says T-inconsistent
- SAT solver detects $\{1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2, \overline{1} \vee \overline{2} \vee \overline{3} \vee 4\}$ UNSAT!

Why "lazy"? Theory information used lazily when checking T-consistency of propositional models

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- Check T-consistency only of full propositional models

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments
- Given a T-inconsistent assignment M, add $\neg M$ as a clause

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments
- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments
- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- Upon a T-inconsistency, add clause and restart

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments
- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- ~~Upon a T-inconsistency, add clause and restart~~
- Upon a T-inconsistency, backtrack to some point where the assignment was still T-consistent

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments
- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- ~~Upon a T-inconsistency, add clause and restart~~
- Upon a T-inconsistency, backtrack to some point where the assignment was still T-consistent
- Boolean engine decides to set which variable to which value

State of the art in SMT: lazy approach . 2

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignments
- ~~Given a T-inconsistent assignment M, add $\neg M$ as a clause~~
- Given a T-inconsistent assignment M, identify a T-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
- ~~Upon a T-inconsistency, add clause and restart~~
- Upon a T-inconsistency, backtrack to some point where the assignment was still T-consistent
- ~~Boolean engine decides to set which variable to which value~~
- Let the theory guide the search by looking for T-consequences

State of the art in SMT: lazy approach . 3

DPLL(T) = DPLL(X) engine + T-Solver

where T-Solver has the following functionalities:

- Determine the T-consistency of a set of literals M
- If M is T-inconsistent, identify a (small) subset $M_0 \subseteq M$ also T-inconsistent [smallest, irredundant]
- Incrementality: if M is augmented with l, checking T-consistency of $M\,l$ must be faster than reprocessing the whole sequence from scratch
- Backtrack: due to DPLL(X) backtrack, solver must support it
- Theory propagation: determine input T-consequences of M
- If $M \models_T l$, identify a (small) explanation $M_0 \subseteq M$ such that also $M_0 \models_T l$ [smallest, irredundant]. Needed for backjump.

Research interests: SAT/SMT solvers

- Barcelogic is our SMT solver: see results in SMT-COMP'05...'09
- DPLL(X) engine is a state-of-the-art competitive SAT solver: see 3rd place in SAT-RACE'08 (1st place for UNSAT instances)
- Theory solvers for:
  - EUF
  - (Integer/Real) Difference Logic
  - Linear (Integer/Real) Arithmetic
  - Arrays and combinations
  - Non-linear Integer Arithmetic
- 1st place in Non-linear Integer Arithmetic in SMT-COMP'09
- **DEVELOPMENT OF EFFICIENT SAT/SMT SOLVERS**

Research interests: bitvectors

- Theory of fixed-size bitvectors:
  $(x\#y)[31:16] = (z\#z)[15:0] \quad \wedge \quad (x+1 = y \ll 2 \quad \vee \quad x = y\&z)$
- Important applications in verification
  - Hardware
  - Software: device drivers, ...
- Can be reduced to modular arithmetic
  - \+ behaves well with arithmetic
  - \- performs poorly if stream/bitwise part is significant
- Current state-of-the-art tools bit-blast and reduce to SAT: complementary situation
- **DESIGN OF THEORY SOLVER TAKING BEST OF BOTH**

Research interests: optimization

- We focus on SAT and SMT problems where models M are sought such that a given cost function $f(M)$ is minimized
- Applications:
  - Min/Max-Ones
  - Max-SAT
- Max-SAT particularly interesting:
  - INPUT: set of pairs $\{(C_1, w_1), \ldots, (C_m, w_m)\}$ where each pair is a clause C_i with its weight w_i.
  - OUTPUT: model M that minimizes the sum of the weights of the clauses false in M
- **DESIGN OF AN EFFICIENT SAT/SMT OPTIMIZER**

Thank you!