The DPLL(T) Approach to Satisfiability Modulo Theories

Albert Oliveras Llunell

Ph.D. Dissertation Defense

Universitat Politècnica de Catalunya

July 7, 2006

Departament de Llenguatges i Sistemes Informàtics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Overview of the talk

- Introduction

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver
  - Difference Logic [CAV'05]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver
  - Difference Logic [CAV'05]
  - Equality with Uninterpreted Functions [LPAR'03, RTA'05]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver
  - Difference Logic [CAV'05]
  - Equality with Uninterpreted Functions [LPAR'03, RTA'05]
- Applications of DPLL(T):

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver
  - Difference Logic [CAV'05]
  - Equality with Uninterpreted Functions [LPAR'03, RTA'05]
- Applications of DPLL(T):
  - Predicate abstraction [CAV'06]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver
  - Difference Logic [CAV'05]
  - Equality with Uninterpreted Functions [LPAR'03, RTA'05]
- Applications of DPLL(T):
  - Predicate abstraction [CAV'06]
  - Optimization problems [SAT'06]

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT) [LPAR'04]
- Satisfiability Modulo Theories (SMT) [LPAR'04]
- DPLL(T) = DPLL(X) + T-Solver [CAV'04, LPAR'05]
- Development of T-Solver
  - Difference Logic [CAV'05]
  - Equality with Uninterpreted Functions [LPAR'03, RTA'05]
- Applications of DPLL(T):
  - Predicate abstraction [CAV'06]
  - Optimization problems [SAT'06]
- Conclusions and future work

Introduction

- Historically, automated reasoning $\equiv$ uniform proof-search procedures for FO logic
- Little 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 (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
  - \- certain (but acceptable) loss of efficiency

GOAL OF THE THESIS:
develop theory, techniques and tools to improve SMT

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

Introduction to SAT

Problem definition:

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

Example:

- $(p \vee q) \quad \wedge \quad (\overline{p} \vee q) \quad \wedge \quad (r \vee \overline{q})$ is SAT with model $\{q, p, r\}$
- $(p \vee q) \quad \wedge \quad (\overline{p} \vee q) \quad \wedge \quad (r \vee \overline{q}) \quad \wedge \quad (\overline{r} \vee \overline{q})$ 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
- DPLL presentations by means of pseudo-code fragments

Contributions to SAT: Abstract DPLL

GOAL: provide abstract presentation of DPLL that

- models conceptual features
- abstracts away implementation details

Our DPLL abstract formulation:

- Transition rules $M \parallel F \;\Rightarrow\; M' \parallel F'$
- On states $M \parallel F$, where:
  - M (a sequence of literals) is the partial assignment being built
  - F is the current set of clauses

Consider the following **Classical DPLL** rules, where literals l in red are decision literals:

Contributions to SAT: Abstract DPLL (2)

Extending the assignment:

UnitPropagate :

$$M \parallel F, C \vee l \quad \Rightarrow \quad M\, l \parallel F, C \vee l \quad \textbf{if} \begin{cases} M \models \neg C \text{ and } l \text{ undefined in } M \end{cases}$$

Decide :

$$M \parallel F \quad \Rightarrow \quad M\, l \parallel F \quad \textbf{if} \begin{cases} l \text{ is undefined in } M \end{cases}$$

Repairing the assignment:

Backtrack :

$$M\, l\, N \parallel F, C \quad \Rightarrow \quad M\, \neg l \parallel F, C \quad \textbf{if} \begin{cases} M\, l\, N \models \neg C \\ N \text{ contains no decision literals} \end{cases}$$

Fail :

$$M \parallel F, C \quad \Rightarrow \quad \mathit{fail} \quad \textbf{if} \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$\emptyset \parallel \quad \overline{1}\lor 2,\;\; \overline{3}\lor 4,\;\; \overline{5}\lor\overline{6},\;\; 6\lor\overline{5}\lor\overline{2} \qquad \Rightarrow \qquad (\text{Decide})$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$\begin{array}{llllll} \emptyset & \parallel & \overline{1}\lor 2,\;\; \overline{3}\lor 4,\;\; \overline{5}\lor\overline{6},\;\; 6\lor\overline{5}\lor\overline{2} & \Rightarrow & (\text{Decide}) \\ 1 & \parallel & \overline{1}\lor 2,\;\; \overline{3}\lor 4,\;\; \overline{5}\lor\overline{6},\;\; 6\lor\overline{5}\lor\overline{2} & \Rightarrow & (\text{UnitPropagate}) \end{array}$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$\begin{array}{rcllll}
\emptyset & \parallel & \overline{1}\lor 2,\; \overline{3}\lor 4,\; \overline{5}\lor\overline{6},\; 6\lor\overline{5}\lor\overline{2} & \Rightarrow & (\text{Decide}) \\
1 & \parallel & \overline{1}\lor 2,\; \overline{3}\lor 4,\; \overline{5}\lor\overline{6},\; 6\lor\overline{5}\lor\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2 & \parallel & \overline{1}\lor 2,\; \overline{3}\lor 4,\; \overline{5}\lor\overline{6},\; 6\lor\overline{5}\lor\overline{2} & \Rightarrow & (\text{Decide})
\end{array}$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$
\begin{array}{rcllllcl}
\emptyset & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1\;2\;3 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate})
\end{array}
$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$
\begin{array}{rcllllcl}
\emptyset & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1\;2\;3 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2\;3\;4 & \parallel & \overline{1}\vee 2, & \overline{3}\vee 4, & \overline{5}\vee\overline{6}, & 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide})
\end{array}
$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$
\begin{array}{rcllll}
\emptyset & \parallel & \overline{1}\lor 2, & \overline{3}\lor 4, & \overline{5}\lor\overline{6}, & 6\lor\overline{5}\lor\overline{2} \quad \Rightarrow \quad (\text{Decide})\\
1 & \parallel & \overline{1}\lor 2, & \overline{3}\lor 4, & \overline{5}\lor\overline{6}, & 6\lor\overline{5}\lor\overline{2} \quad \Rightarrow \quad (\text{UnitPropagate})\\
1\;2 & \parallel & \overline{1}\lor 2, & \overline{3}\lor 4, & \overline{5}\lor\overline{6}, & 6\lor\overline{5}\lor\overline{2} \quad \Rightarrow \quad (\text{Decide})\\
1\;2\;3 & \parallel & \overline{1}\lor 2, & \overline{3}\lor 4, & \overline{5}\lor\overline{6}, & 6\lor\overline{5}\lor\overline{2} \quad \Rightarrow \quad (\text{UnitPropagate})\\
1\;2\;3\;4 & \parallel & \overline{1}\lor 2, & \overline{3}\lor 4, & \overline{5}\lor\overline{6}, & 6\lor\overline{5}\lor\overline{2} \quad \Rightarrow \quad (\text{Decide})\\
1\;2\;3\;4\;5 & \parallel & \overline{1}\lor 2, & \overline{3}\lor 4, & \overline{5}\lor\overline{6}, & 6\lor\overline{5}\lor\overline{2} \quad \Rightarrow \quad (\text{UnitPropagate})
\end{array}
$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$
where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$
\begin{array}{rclcl}
\emptyset & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide})\\
1 & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate})\\
1\ 2 & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide})\\
1\ 2\ 3 & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate})\\
1\ 2\ 3\ 4 & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide})\\
1\ 2\ 3\ 4\ 5 & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate})\\
1\ 2\ 3\ 4\ 5\ \overline{6} & \parallel & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Backtrack})
\end{array}
$$

Contributions to SAT: Abstract DPLL (3)

A DPLL procedure for F is any derivation: $\emptyset \parallel F \;\Rightarrow\; \ldots \;\Rightarrow\; S$ where S is a final state (no rule applies). It always terminates.

- One can easily prove that, if the final state S is:
  - *fail* then F is unsat.
  - of the form $M \parallel F$ then M is a model
- Example:

$$
\begin{array}{rclcl}
\emptyset & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1 & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2 & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1\;2\;3 & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2\;3\;4 & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
1\;2\;3\;4\;5 & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2\;3\;4\;5\;\overline{6} & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Backtrack}) \\
1\;2\;3\;4\;\overline{5} & \parallel & \overline{1}\vee 2,\;\; \overline{3}\vee 4,\;\; \overline{5}\vee\overline{6},\;\; 6\vee\overline{5}\vee\overline{2} & \Rightarrow & \ldots
\end{array}
$$

Contributions to SAT: Abstract DPLL (4)

Abstract DPLL can be extended to deal with modern DPLL features:

- Backjump: (generalizes Backtrack)

$$M\ l\ N \parallel F, C \ \Rightarrow\ M\ l' \parallel F, C \quad \textbf{if} \begin{cases} M\ l\ N \models \neg C, \text{ and } \exists\, C' \vee l' \text{ s.t.:} \\ \text{(i)} F, C \models C' \vee l', \text{(ii) } M \models \neg C' \\ \text{(iii) } l' \text{ undefined in } M \end{cases}$$

- Learn: (learning backjump clauses avoids "similar" conflicts)

$$M \parallel F \ \Rightarrow\ M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F \\ F \models C \end{cases}$$

- Forget: (removes "inactive" clauses)

$$M \parallel F, C \ \Rightarrow\ M \parallel F \quad \textbf{if} \left\{ F \models C \right.$$

Abstract DPLL provides formal and uniform proofs of correctness and completeness of many variants, strategies and extensions.

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

Introduction to SMT

- Some problems are more naturally expressed in other logics than propositional logic, e.g:
  - Software verification needs reasoning about equality, arithmetic, data structures, ...
- SMT consists of deciding the satisfiability of a (ground) FO formula with respect to a background theory
- Example (Equality with Uninterpreted Functions – EUF):

$$g(a)=c \quad \wedge \quad (\ f(g(a)) \neq f(c) \ \vee \ g(a)=d\ ) \quad \wedge \quad c \neq d$$

- Wide range of applications:
  - Predicate abstraction
  - Model checking
  - Equivalence checking
  - Static analysis
  - Scheduling
  - ...

State of the art in SMT: eager approach

- Methodology: translate problem into equisatisfiable propositional formula and use off-the-shelf SAT solver [Bryant, Velev, Pnueli, Lahiri, Seshia, Strichman, ...]
- Why "eager"?
  Search uses all theory information from the beginning
- Characteristics:
  + Can use best available SAT solver
  - Sophisticated encodings are needed for each theory
- Tools: UCLID [Lahiri, Seshia and Bryant]

State of the art in SMT: lazy approach

Methodology:

Example: consider EUF and

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

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

State of the art in SMT: lazy approach

Methodology:

Example: consider EUF and

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

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

State of the art in SMT: lazy approach

Methodology:

Example: consider EUF and

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

- 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

Methodology:

Example: consider EUF and

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

- 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

Methodology:

Example: consider EUF and

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

- 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

UPC
Departament de Llenguatges i Sistemes Informàtics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

State of the art in SMT: lazy approach

Methodology:

Example: consider EUF and

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

- 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 $\{1,\ \overline{2}\vee 3,\ \overline{4},\ \overline{1}\vee 2,\ \overline{1}\vee\overline{2}\vee\overline{3}\vee 4\}$
UNSATISFIABLE

State of the art in SMT: lazy approach (2)

- Why “lazy”?
  Theory information used lazily when checking T-consistency of propositional models
- Characteristics:
  - + Modular and flexible
  - - Theory information does not guide the search
- Tools: CVC-Lite, ICS, MathSAT, TSAT+, Verifun, ...

State of the art in SMT: lazy approach (3)

Several optimizations for enhancing efficiency:

- Check T-consistency only of full propositional models

State of the art in SMT: lazy approach (3)

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

State of the art in SMT: lazy approach (3)

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- Given a T-inconsistent assignment M, add $\neg M$ as a clause

State of the art in SMT: lazy approach (3)

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- ~~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 (3)

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- ~~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 (3)

Several optimizations for enhancing efficiency:

- ~~Check T-consistency only of full propositional models~~
- Check T-consistency of partial assignment while being built

- ~~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, bactrack to some point where the assignment was still T-consistent

Contributions to SMT: T-Abstract DPLL

GOAL: provide abstract presentation of the lazy approach that

- provides a clean and precise description of various strategies
- facilitates their understanding and comparison

Rules are easily extended to accomodate theory reasoning, e.g.:

- T-Learn:

$$M \parallel F \quad \Rightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F \\ F \models_T C \end{cases}$$

- T-Backjump:

$$M\ l\ N \parallel F, C \Rightarrow M\ l' \parallel F, C \quad \textbf{if} \begin{cases} M\ l\ N \models \neg C, \text{ and } \exists\ C' \vee l' \text{ s.t.:} \\ \text{(i)} F, C \models_T C' \vee l', \text{(ii) } M \models \neg C' \\ \text{(iii) } l' \text{ undefined in } M \end{cases}$$

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

Introduction to DPLL(T)

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

- Modular and flexible, as CLP(X) in Constraint Logic Progr.
- More Pruning: due to Theory Propagation:

T-Propagate :

$$M \parallel F \quad \Rightarrow \quad M\, l \parallel F \quad \textbf{if} \begin{cases} M \models_T l \\ l \text{ or } \neg l \text{ occurs in } F \text{ and not in } M \end{cases}$$

- Search guided by T-Solver by finding T-consequences,
  instead of only validating it as in lazy approach.
- [Armando et al]: Add $\neg l$. If T-inconsistent then infer l.
  But Theory Propagation in DPLL(T):
  -T-Solvers specialized and fast in it.
  -fully exploited in conflict analysis (non-trivial)

Introduction to DPLL(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{UnitPropagate})$$

Introduction to DPLL(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{UnitPropagate})$$

$$1 \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{T-Propagate})$$

Introduction to DPLL(T) - Example

Consider again EUF and the formula:

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

$$\begin{array}{rclcl}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\text{T-Propagate}) \\
1\ \overline{2} & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\text{UnitPropagate})
\end{array}$$

UPC
Departament de Llenguatges i Sistemes Informàtics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Introduction to DPLL(T) - Example

Consider again EUF and the formula:

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

$$
\begin{array}{rclcl}
\emptyset & \| & 1,\;\; \overline{2}\vee 3,\;\; \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
1 & \| & 1,\;\; \overline{2}\vee 3,\;\; \overline{4} & \Rightarrow & (\text{T-Propagate}) \\
1\;2 & \| & 1,\;\; \overline{2}\vee 3,\;\; \overline{4} & \Rightarrow & (\text{UnitPropagate}) \\
1\;2\;3 & \| & 1,\;\; \overline{2}\vee 3,\;\; \overline{4} & \Rightarrow & (\text{T-Propagate})
\end{array}
$$

Introduction to DPLL(T) - Example

Consider again EUF and the formula:

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

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \\
1\ \overline{2} & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\
1\ \overline{2}\ 3 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \\
1\ \overline{2}\ 3\ 4 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{Fail})
\end{array}$$

Introduction to DPLL(T) - Example

Consider again EUF and the formula:

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

$$\begin{array}{rclcl}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\textsf{UnitPropagate}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\textsf{T-Propagate}) \\
1\ 2 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\textsf{UnitPropagate}) \\
1\ 2\ 3 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\textsf{T-Propagate}) \\
1\ 2\ 3\ 4 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow & (\textsf{Fail}) \\
 & \mathit{fail} & & &
\end{array}$$

Contributions of DPLL(T): our system

- BarcelogicTools for SMT is the DPLL(T) implementation developed as part of this thesis
- Relatively small system (10k lines of C code)
- DPLL(X) engine: two-watched literals, 1UIP learning scheme, VSIDS-like decision heuristics,...
- Theory solvers for:
  - Difference Logic (DL):
    - fragment of linear arithmetic, atoms are bounds on difference of variables, e.g. $a - b \leq 3$
    - useful for timing constraints
    - 90% of arithmetic constraints in verification are DL
  - Equality with Uninterpreted Functions (EUF)

Contributions of DPLL(T): SMT-COMP'05

	top-3 systems	# Pbs solv.	Time (secs.)
	BarcelogicTools	39	8358
EUF (50 pbs.):	Yices	37	9601
	MathSAT	33	12386
	BarcelogicTools	41	6341
RDL (50):	Yices	37	9668
	MathSAT	37	10408
	BarcelogicTools	47	3531
IDL (51):	Yices	47	4283
	MathSAT	46	4295
	BarcelogicTools	45	2705
UFIDL (49):	Yices	36	9789
	MathSAT	22	17255

Other tools:

- CVC-Lite (Barrett)
- Ario (Sakallah)
- Sateen (Somenzi)
- ...

Time outs are counted as 600s.

Contributions of DPLL(T): T-propagation

	BarcelogicTools setting	# Pbs solved	Time (secs.)
EUF (152 pbs.):	Theory propagation	118	27423
	No theory propagation	101	36134
RDL (170 pbs.):	Theory propagation	112	37923
	No theory propagation	99	51590
IDL (309 pbs.):	Theory propagation	298	9376
	No theory propagation	293	12668
UFIDL (280 pbs.):	Theory propagation	278	3354
	No theory propagation	190	70264

Time outs are counted as 600s.

- T-Propagate prune of the search space is expected a priori
- But T-Propagate impact on runtime is also very positive
- However, care must be taken when implementing it

Contributions of DPLL(T): T-propagation

Extension of conflict analysis mechanisms.
For finding backjump clauses, each l is in M due to:

- Decide
- UnitPropagate with clause C: resolve with C
- T-Propagate: resolve with (small) appropriate explanation $l_1 \land \ldots \land l_n \rightarrow l$ provided by T-Solver

How should it be implemented?

- UnitPropagate: store a pointer to clause C, as in SAT-solvers
- T-Propagate: should we compute and store the explanation?
  – If possible, compute explanations only on demand, since some of them may never be needed
  – Example: on the SMT-COMP benchmarks, on average only one Explain for every 250 T-Propagate.

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- **Development of T-Solver**
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

Introduction to T-Solver development

What does DPLL(T) need from a T-Solver?

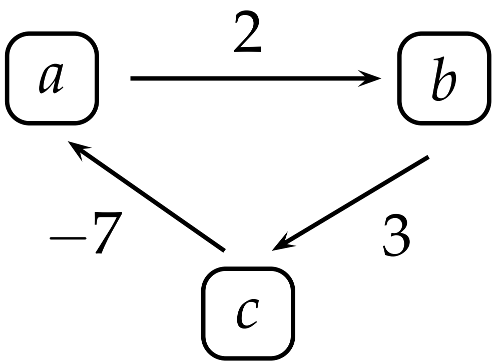
- Determine the T-consistency of a set of literals M
- If M is T-inconsistent, identify a (small) subset $M_0 \subseteq M$ also T-inconsistent [smallest, irredundant]
- Incrementality: if M is augmented with l, checking T-consistency of $M\,l$ must be faster than reprocessing the whole sequence from 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.

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

State of the art in DL

- Given $M = \{a{-}b \leq 2,\ b{-}c \leq 3,\ c{-}a \leq -7\}$, construct weighted graph $\mathcal{G}(M)$



- M is T-inconsistent iff $\mathcal{G}(M)$ has a negative cycle
- Bellmann-Ford-like algorithms to find such cycles in $O(nm)$
- Irredundant inconsistent subsets are negative cycles
- Theory propagation was not considered

T-Solver development:contributions (DL)

- Key idea: exhaustive theory propagation avoids consistency checks: $M\,l$ is T-inconsistent iff $M \models_T \neg l$. Hence we would have added $\neg l$ right after M.
- For detecting all consequences of a new literal $a - b \leq k$:

  $c - d \leq k_1$ is T-entailed iff there is a path form c to d with length at most k_1. Hence *T-Solver* checks all shortest paths

  $$c \xrightarrow{k'} * \quad a \xrightarrow{k} b \xrightarrow{k''} * \quad d$$

  and finds all input literals entailed by $c - d \leq k' + k + k''$
- Complexity: $O(nm + N)$, being N the number of input literals
- Irredundant explanations for $c - d \leq k$ given by the shortest path from c to d

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - EUF
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

State of the art in EUF

- Consistency of sets of ground (dis)equations determined by congruence closure procedures: extension of union-find enforcing functional consistency
- Best known time is $O(n \ \log n)$
- Classical procedures:
  - Shostak'78
  - Nelson-Oppen'80
  - Downey-Sethi-Tarjan'80
- Modern (abstract) presentations:
  - Kapur'97
  - Bachmair-Tiwari'00
- No support for theory propagation or explanation generation

T-Solver development:contribut'ns (EUF)

- New fast and simple $O(n \ \log n)$ algorithm (adopted by others, e.g. MathSAT)
- After initial transformations (done once and for all) only one binary function symbol f. All equalities are either
  - $f(a,b)=c$, or
  - $a=b$
- Regarding theory propagation:
  - Computes all entailed equalities
  - Misses some entailed disequalities, e.g:

$$\left.\begin{array}{lll} f(a,b)=k & f(c,b)=l & \\ f(d,b)=a & f(b,d)=c & k\neq l \end{array}\right\} \models_T b\neq d$$

- Extension to deal in $O(n \ \log n)$ with integer offsets, i.e. atoms of the form $f(a-2,c)+2=g(c,h(a)+2)$

T-Solver for EUF: contributions

Production of explanations:

- Starting point is a proof-producing Union-Find (UF)
- Already non-trivial for UF, since information is lost if paths are compressed
- Extension to proof-producing CC:
  - Processes n equations in $O(n \ \log n)$ time
  - Explain of size k in almost optimal $O(k \ \alpha(k)\ )$ time
- Quality of explanations
  - Find the smallest explanation is NP-hard
  - Our explanations may contain some redundant steps
  - However, only 3% of explains are redundant in practice
  - An almost exact test detects possibly redundant explanations than can be later post-processed if wanted

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

Introduction to predicate abstraction

Predicate abstraction:

- Technique for constructing finite-state abstractions from infinite-state systems
- Resulting abstraction efficiently analyzed using Boolean techniques
- Applications to verification of software, hardware and high-level protocols

Key operation:

- INPUT: a formula φ and set of predicates P (built over the signature of a theory T)
- OUTPUT: the most precise approximation of φ using P, either $\mathcal{F}_P(\varphi)$: weakest formula over P T-entailing φ or $\mathcal{G}_P(\varphi)$: strongest formula over P T-entailed by φ.

State of the art in predicate abstraction

Two importants facts:

- $\mathcal{F}_P(\varphi)$ is $\neg\mathcal{G}_P(\neg\varphi)$.
- $\mathcal{G}_P(\varphi)$ is $\bigvee\{c \mid c$ is a minterm over P and $c \wedge \varphi$ is T-satisfiable$\}$

Three main approaches (in chronological order):

- Check satisfiability of $c \wedge \varphi$ for all minterms c (exponential number of calls) [Saidi & Shankar, Das & Dill & Park,...]
- Reduce the problem to Boolean quantifier elimination (and use SAT-solving techniques) [Lahiri & Bryant,Clarke & al,...]
- Use symbolic decision procedures (may avoid exponential work) [Lahiri & Ball & Cook]

Contributions to predicate abstraction

Our computation of an approximation of φ is based on

- Using an SMT solver to compute all models over P of φ
- Using a BDD package to store all models
- Extracting from the BDD a compact representation of the approximation

Key points for the success:

- Careful enumeration of models (All-SAT techniques)
- Incrementality of the approach
- Use of our state-of-the-art DPLL(T) implementation

Another contribution:

- Proposed a method for refining predicate abstractions with a fixed set of predicates

Contributions to predicate abstraction (2)

- Microsoft SLAM project: Windows device drivers verification
- Initially, theorem prover ZAP [Ball et al, CAV'04] was used for predicate abstraction
- Specialized Symbolic Decision Procedures (SDP) [Lahiri et al, CAV'05] obtained 100x speedup factor
- The biggest availabe set of benchmarks (around 700 queries) processed by SDP in 700 seconds
- BarcelogicTools only took 5 seconds, another 100x speedup

Contributions to predicate abstraction (3)

Benchmark family	UCLID time (secs.)	BCLT time (secs.)	BCLT speedup
UCLID Suite:			
aodv	657	4.6	143x
bakery	245	11	22x
BRP	3.5	0.1	35x
cache_ibm	34	1.3	26x
cache_bounded	1119	23	49x
DLX	335	13	26x
OOO	921	36	26x

Hardware and protocol verification benchmarks used in [Lahiri and Bryant, CAV'04]

Again, times are in seconds.

Contributions to predicate abstraction (3)

Benchmark family	UCLID time (secs.)	BCLT time (secs.)	BCLT speedup
Rec. Data Struct.:			
reverse_acyclic	20	0.6	33x
set_union	22	0.7	31x
simple_cyclic	3.7	0.11	34x
sorted_int	765	19	40x

Benchmarks from the verification of programs manipulating linked lists (Qaader and Lahiri, POPL'06)

Again, times are in seconds.

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- Conclusions and future work

Introduction to optimization problems

- We focus on SAT and SMT problems where models M are sought such that a given cost function $f(M)$ is minimized
- Applications:
  - Min-Ones
  - Min-Max metrics
  - Max-SAT
- Max-SAT particularly interesting and well-studied:
  - 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.
  - OUPUT: model M that minimizes the sum of the weights of the clauses false in M

State of the art in optimization problems

Two types of tools:

- Pseudo-boolean solvers: PBS, Pueblo, MiniSAT+,...
  - Good when optimization is not a hard part (not many models)
  - Inherit learning and heuristics from SAT
  - Can deal with large but easy problems
- Specialized tools for, e.g. Max-SAT: ToolBar, MAX-Solver, ...
  - Good when optimization is the hard part (lots of models)
  - No learning, specialized heuristics and pruning rules
  - Can deal with small but challenging problems

Contributions to optimization problems

- Our implementation of branch and bound in DPLL(T):
  - the theory T knows about (i) the cost function f and (ii) the cost of best solution so far
  - after a model with cost C is found, we strengthen the theory T by making T-inconsistent all models of cost greater or equal to C

Its application to Max-SAT:

- Similar in spirit to Pseudo-Boolean solvers
- Incorporates specialized deduction rules for Max-SAT, e.g: upon detection of unit weighted clauses (l, w_1) and $(\neg l, w_2)$ then $min(w_1, w_2)$ can be added to the current cost
- Specialized rules need to be further worked out

Contributions to optimization pbms. (2)

Its application to Max-SMT:

- Modeled and solved well-known hard Radio Link Frequency Assignment Problems
- Radio frequencies have to be assigned satisfying some distance constraints (Difference Logic)
- Although no specialized heuristics were used, positive comparison with best weighted CSP solver (Toolbar):

Benchmark name	Toolbar			DPLL(T)
	Static	Default	Jeroslow	
SUBCELAR_6_0	0.2	0.4	0.2	5
SUBCELAR_6_1	85	252	65	90
SUBCELAR_6_2	127	982	25	132
SUBCELAR_6_3	7249	2169	355	636
SUBCELAR_6_4	7021	$>$ 5 hours	1942	1417

Overview of the talk

- Introduction
- Propositional Satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- DPLL(T) = DPLL(X) + T-Solver
- Development of T-Solver
  - Difference Logic
  - Equality with Uninterpreted Functions
- Applications of DPLL(T):
  - Predicate abstraction
  - Optimization problems
- **Conclusions and future work**

Conclusions

- Developed an abstract framework to model DPLL-based solvers for SAT and SMT
- Introduced the DPLL(T) approach to SMT
  - Careful study of theory propagation and its integration in conflict analysis
  - Developed BarcelogicTools, a very efficient implementation
- Designed efficient proof-producing solvers for DL and EUF
- Applied DPLL(T) to predicate abstraction and optimization problems

Future work

- Move to a DPLL($T_1 \ldots T_n$) Nelson-Oppen-based implementation
- Adding support for quantifiers
- Extend system for verification purposes
- Non-standard applications of SMT: e.g. classification of finite mathematical structures

Publications

1. R. Nieuwenhuis, A. Oliveras. Congruence Closure with Integer Offsets. LPAR'03.
2. H. Ganzinger, G. Hagen, R. Nieuwenhuis , A. Oliveras, C. Tinelli. DPLL(T): Fast Decision Procedures. CAV'04.
3. R. Nieuwenhuis, A. Oliveras, C. Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. LPAR'04.
4. R. Nieuwenhuis, A. Oliveras. Proof-Producing Congruence Closure. RTA'05.
5. R. Nieuwenhuis, A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. CAV'05.
6. R. Nieuwenhuis, A. Oliveras. Decision Procedures for SAT, SAT Modulo Theories and Beyond. LPAR'05.
7. S. K. Lahiri, R. Nieuwenhuis, A. Oliveras. SMT Techniques for Fast Predicate Abstraction. CAV'06.
8. R. Nieuwenhuis, A. Oliveras. On SAT Modulo Theories and Optimization Problems. SAT'06.
9. R. Nieuwenhuis, A. Oliveras, C. Tinelli. Solving SAT and SAT Modulo Theories . JACM, accepted with minor modifications (3rd review round).
10. R. Nieuwenhuis, A. Oliveras. Proof-producing Congruence Closure. Information and Computation, accepted with minor modifications (2nd review round).