Abstract DPLL and Abstract DPLL Modulo Theories

Robert Nieuwenhuis[1], Albert Oliveras[1], and Cesare Tinelli[2]

[1] Technical University of Catalonia
[2] The University of Iowa

Overview of the talk

- Motivation: SAT and SMT
- Propositional case
  - The Basic DPLL System
  - The DPLL System
- SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory propagation

Propositional satisfiability: SAT

- Deciding the satisfiability of a propositional formula is a very important problem
- Theoretical interest: first established NP-Complete problem, phase transition, ...
- Practical interest: applications to scheduling, planning, logic synthesis, verification,...
  - Successful procedure: DPLL + backumping + learning

Satisfiablity Modulo Theories

- Some problems are more naturally expressed in other logics
  - Pipelined microprocessors: logic EUF, atoms are $f(g(a,b),c) = g(c,a)$
  - Timed automata: separation logic, atoms are $a < b + 2$
  - Software verification: combination of theories, e.g. $5 + car(a+2) = cdr(a+1)$
- Deciding the satisfiability of a (ground) formula with respect to a background theory has lots of applications (SMT problem)

Lifting SAT to SMT

- Eager approach: obtain an equisatisfiable propositional formula and use a SAT solver (UCLID)
- Lazy approach: abstract the formula into a propositional one and use a theory decision procedure to refine it (CVC, ICS, MathSAT, TSAT++, ...)
- DPLL(T): smarter way to use the theory information

Overview of the talk

- Motivation: SAT and SMT
- Propositional case
  - The Basic DPLL System
  - The DPLL System
- SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory propagation

The Basic DPLL Procedure

- Tries to incrementally build a model M for the CNF formula F.
- M is augmented by deciding a literal or deducing one from M and F.
- When a wrong decision is detected, the procedure backtracks.

We will model it with a transition system between states:

$$M \parallel F \Longrightarrow M' \parallel F'$$

The Basic DPLL System

Extending the model:

UnitProp

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

Decide

$$M \parallel F \implies M\, l^{\mathsf{d}} \parallel F \;\; \textbf{if} \begin{cases} l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

The Basic DPLL System

Repairing the model:

Fail

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

Backjump

$$M\, l^{\mathsf{d}}\, N \parallel F \implies M\, l' \parallel F \quad \textbf{if} \begin{cases} \text{for some clause } C \vee l' : \\ \quad F \models C \vee l' \text{ and } M \models \neg C \\ \quad l' \text{ is undefined in } M \\ \quad l' \text{ or } \neg l' \text{ occurs in } F \end{cases}$$

Basic DPLL System - Example

$$\begin{array}{rcll}
\emptyset & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{Decide}) \\
1 & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2 & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{Decide}) \\
1\ 2\ 3 & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3\ 4 & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{Decide}) \\
1\ 2\ 3\ 4\ 5 & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad (\text{Backjump}) \\
1\ 2\ \overline{5} & \| & \overline{1} \lor 2,\ \ \overline{3} \lor 4,\ \ \overline{5} \lor \overline{6},\ \ 6 \lor \overline{5} \lor \overline{2} & \Longrightarrow \quad \ldots
\end{array}$$

Basic DPLL System - Example

$$\ldots$$

$$1\ 2\ 3\ 4\ 5\ \overline{6} \ \| \ \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} \quad \Longrightarrow \quad \text{(Backjump)}$$

$$1\ 2\ \overline{5} \ \| \ \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$$

In this case $F \models \overline{1}\vee\overline{5}$ we have by resolution

$$\dfrac{\dfrac{\overline{1}\vee 2 \quad 6\vee\overline{5}\vee\overline{2}}{\overline{1}\vee 6\vee\overline{5}} \qquad \overline{5}\vee\overline{6}}{\overline{1}\vee\overline{5}}$$

and before deciding 3, we could have deduced $\overline{5}$.

Basic DPLL System-Correctness

- $\emptyset \parallel F \Longrightarrow^! \mathit{fail}$ iff F is unsatisfiable
- $\emptyset \parallel F \Longrightarrow^! M \parallel F$ iff F is satisfiable

Key ingredients:

- All rules decrease with respect to a well-founded ordering between states
- When M falsifies a clause in F, either Fail or Backjump apply.

The DPLL System

Learning and forgetting clauses:

Learn

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

Forget

$$M \parallel F, C \implies M \parallel F \;\; \textbf{if} \;\; F \models C$$

The DPLL system terminates if no clause is learned/forgotten infinitely often

The DPLL system - Strategies

- Applying one rule of the Basic DPLL system between each two Learn ensures termination
- In practice, Learn is usually (but not only) applied right after Backjump.
- A common strategy is to apply the rules using the following priorities:
  1. If there is a clause in F which is false in M apply Fail or Backjump + Learn
  2. Apply UnitProp
  3. Apply Decide

Overview of the talk

- Motivation: SAT and SMT
- Propositional case
  - The Basic DPLL System
  - The DPLL System
- SMT case
  - Very Lazy Theory Learning
  - Lazy Theory Learning
  - Theory propagation

Very Lazy Approach for SMT

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

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

Very Lazy Approach for SMT

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

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

Very Lazy Approach for SMT

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

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

Very Lazy Approach for SMT

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

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

Very Lazy Approach for SMT

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

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

Very Lazy Approach for SMT

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

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

Very Lazy Approach - Modelling

- The process within the SAT solver is modelled using the DPLL sytem
- The interaction between the theory solver and the SAT solver is modelled with the rule

Very Lazy Theory Learning

$$M\,l\,M_1 \parallel F \implies \emptyset \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \vee \overline{l} \ \ \textbf{if} \begin{cases} M\,l\,M_1 \models F \\ \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \ldots \wedge l_n \models_T \overline{l} \end{cases}$$

Lazy approach

- Detects T-inconsistent partial models using

Lazy Theory Learning

$$M\, l\, M_1 \parallel F \Longrightarrow M\, l\, M_1 \parallel F, \overline{l_1} \vee \ldots \vee \overline{l_n} \vee \bar{l} \ \textbf{if} \begin{cases} \{l_1, \ldots, l_n\} \subseteq M \\ l_1 \wedge \ldots \wedge l_n \models_T \bar{l} \\ \bar{l}_1 \vee \ldots \vee \bar{l}_n \vee l \notin F \end{cases}$$

- The learnt clause is false in $M\, l\, M_1$ and hence either Backjump or Fail apply

Lazy approach - Strategies

- A common strategy is to apply the rules using the following priorities:
  1. If there is a clause in F which is false in M apply Fail or Backjump + Learn
  2. If the model is T-inconsistent apply Lazy Theory Learning + (Backjump or Fail)
  3. Apply UnitProp
  4. Apply Decide

DPLL(T) - Eager T-Propagation

- Use the theory information as soon as possible by eagerly applying

Theory Propagate

$$M \parallel F \implies M\,l \parallel F \ \ \textbf{if} \begin{cases} M \models_T l \\ l \text{ or } \bar{l} \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

Eager T-Propagation - Example

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

$$\begin{array}{rcllcl}
\emptyset & \| & 1, & \overline{2}\vee 3, & \overline{4} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1, & \overline{2}\vee 3, & \overline{4} & \Longrightarrow \quad (\text{Theory Propagate}) \\
1\ 2 & \| & 1, & \overline{2}\vee 3, & \overline{4} & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3 & \| & 1, & \overline{2}\vee 3, & \overline{4} & \Longrightarrow \quad (\text{Theory Propagate}) \\
1\ 2\ 3\ 4 & \| & 1, & \overline{2}\vee 3, & \overline{4} & \Longrightarrow \quad (\text{Fail}) \\
 & \mathit{fail} & & & &
\end{array}$$

Eager Theory Propagation

- By eagerly applying Theory Propagate any M will be T-consistent, since $M_1\, l$ is T-inconsistent iff $M_1 \models_T \bar{l}$
- Therefore, Lazy Theory Learning will never apply
- For some logics, e.g. separation logic, this approach is extremely effective
- For some other, e.g. EUF, it is too expensive to detect all T-consequences

Non-Exhaustive T-Propagation

- If Theory Propagate is not eagerly applied, Lazy Theory Learning is needed to repair T-inconsistent models
- The six rules of the DPLL system plus Theory Propagate and Lazy Theory Learning provide a decision procedure for SMT
- Termination is usually ensured this way:
  - Between two Learn applications some rule of the Basic DPLL is applied
  - Apply Backjump or Fail immediately after Lazy Theory Learning

Conclusions

- The DPLL procedure can be modelled in an abstract way
- Modern features such as backjumping, learning (also restarts) can be captured with our transition systems
- Extensions to SMT are possible
- It allows one to describe the strategies of concrete systems in a clean way