SMT Theory and DPLL(T)

Albert Oliveras
Technical University of Catalonia (BarcelonaTech)

Third International SAT/SMT Solver Summer School 2013
Aalto University, Finland
July 4th, 2013

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- History of SMT
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Introduction

- Historically, automated reasoning $\equiv$ uniform proof-search procedures for FO logic
- Limited success: is FO logic the best compromise between expressivity and efficiency?
- Current trend [Sha02] 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 involved encodings
- SMT: propositional logic + domain-specific reasoning
  - \+ improves the expressivity
  - \- certain (but acceptable) loss of efficiency

GOAL OF THIS TALK: introduce SMT, with its main techniques

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- History of SMT
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Need and Applications of SMT

- Some problems are more naturally expressed in other logics than propositional logic, e.g:
  - Software verification needs reasoning about equality, arithmetic, data structures, pointers, functions calls, ...
- 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 [LNO06]
  - Model checking[AMP06]
  - Scheduling [BNO^{+}08b]
  - Test generation[TdH08]
  - ...

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- History of SMT
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Theories of Interest - EUF [BD94, NO80, NO07]

- Equality with Uninterpreted Functions, i.e. "$=$" is equality
- If background logic is FO with equality, EUF is empty theory
- Consider formula

$$a * (f(b) + f(c)) = d \quad \wedge \quad b * (f(a) + f(c)) \neq d \quad \wedge \quad a = b$$

Theories of Interest - EUF [BD94, NO80, NO07]

- Equality with Uninterpreted Functions, i.e. "$=$" is equality
- If background logic is FO with equality, EUF is empty theory
- Consider formula

$$a * (f(b) + f(c)) = d \quad \wedge \quad b * (f(a) + f(c)) \neq d \quad \wedge \quad a = b$$

- Formula is UNSAT, but no arithmetic resoning is needed
- If we abstract the formula into
  $h(a,\ g(f(b), f(c))\ ) = d \quad \wedge \quad h(b,\ g(f(a), f(c))\ ) \neq d \quad \wedge \quad a = b$
  it is still UNSAT
- EUF is used to abstract non-supported constructions, e.g:
  - Non-linear multiplication
  - ALUs in circuits

Theories of Interest - Arithmetic

- Very useful for obvious reasons
- Restricted fragments support more efficient methods:
  - Bounds: $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - Difference logic: $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ [NO05, WIGG05, SM06]
  - UTVPI: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ [LM05]
  - Linear arithmetic, e.g: $2x - 3y + 4z \leq 5$ [DdM06]
  - Non-linear arithmetic, e.g: $2xy + 4xz^2 - 5y \leq 10$ [BLNM+09, ZM10]
  - Variables are either reals or integers

Th. of Int.- Arrays[SBDL01, BNO$^+$08a, dMB09]

- Two interpreted function symbols *read* and *write*
- Theory is axiomatized by:
  - $\forall a \forall i \forall v\ (read(write(a,i,v),i) = v)$
  - $\forall a \forall i \forall j \forall v\ (i \neq j \rightarrow read(write(a,i,v),j) = read(a,j))$
- Sometimes extensionality is added:
  - $\forall a \forall b\ (\ (\forall i (read(a,i) = read(b,i))\ ) \rightarrow a = b$
- Is the following set of literals satisfiable?

$$\begin{array}{ccc} write(a,i,x) \neq b & read(b,i) = y & read(write(b,i,x),j) = y \\ a = b & & i = j \end{array}$$

- Used for:
  - Software verification
  - Hardware verification (memories)

Th. of Interest - Bit vectors [BCF+07, BB09]

- Constants represent vectors of bits
- Useful both for hardware and software verification
- Different type of operations:
  - String-like operations: concat, extract, ...
  - Logical operations: bit-wise not, or, and, ...
  - Arithmetic operations: add, substract, multiply, ...
- Assume bit-vectors have size 3. Is the formula SAT?

$$a[0:1] \neq b[0:1] \;\wedge\; (a|b) = c \;\wedge\; c[0] = 0 \;\wedge\; a[1] + b[1] = 0$$

Combina. of theories [NO79, Sho84, BBC$^+$05]

- In practice, theories are not isolated
- Software verifications needs arithmetic, arrays, bitvectors, ...
- Formulas of the following form usually arise:

$$a = b+2 \land A = write(B, a+1, 4) \land (read(A, b+3) = 2 \lor f(a-1) \neq f(b+1))$$

- The goal is to combine decision procedures for each theory

SMT in Practice

GOOD NEWS: efficient decision procedures for sets of ground literals exist for various theories of interest

PROBLEM: in practice, we need to deal with:

(1) arbitrary Boolean combinations of literals ($\wedge, \vee, \neg$)
(DNF conversion is not a solution in practice)

(2) multiple theories

(3) quantifiers

We will only focus on (1) and (2), but techniques for (3) exist.

SMT in Practice (2)

- SMT-LIB: language, benchmarks, tutorials, ...
- SMT-COMP: performance and capabilities of tools
- SMT Workshop: held annually, collocated with CADE, CAV, SAT.
- Papers at SAT, CADE, CAV, FMCAD, TACAS, ...

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- History of SMT
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

SMT Prehistory - Late 70's and 80's

- Pioneers:
  - R. Boyer, J. Moore, G. Nelson, D. Open, R. Shostak
- Influential results:
  - Nelson-Oppen congruence closure procedure [NO80]
  - Nelson-Oppen combination method [NO79]
  - Shostak combination method [Sho84]
- Influential systems:
  - Nqthm prover [BM90] [Boyer, Moore]
  - Simplify [DNS05] [Detlefs, Nelson, Saxe]

Beginnings of SMT - Early 2000s

KEY FACT: SAT solvers improved performance

Two ways of exploiting this fact:

- Eager approach: encode SMT into SAT

  [Bryant, Lahiri, Pnueli, Seshia, Strichman, Velev, ...]

  [PRSS99, SSB02, SLB03, BGV01, BV02]

  First systems: UCLID [LS04]

- Lazy approach: plug SAT solver with a decision procedure

  [Armando, Barrett, Castellini, Cimatti, Dill, Giunchiglia, deMoura, Ruess, Sebastiani, Stump,...]

  [ACG00, dMR02, BDS02a, ABC$^+$02]

  First systems: TSAT [ACG00], ICS [FORS01], CVC [BDS02b], MathSAT [ABC$^+$02]

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Eager approach

- Methodology: translate problem into equisatisfiable propositional formula and use off-the-shelf SAT solver
- 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, Beaver, Boolector, STP, SONOLAR, Spear, SWORD

Eager approach – Example

Let us consider an EUF formula:

- First step: remove function/predicate symbols.
  Assume we have terms $f(a)$, $f(b)$ and $f(c)$.
  - Ackermann reduction:
    - Replace them by fresh constants A, B and C
    - Add clauses:
      $a = b \;\rightarrow\; A = B$
      $a = c \;\rightarrow\; A = C$
      $b = c \;\rightarrow\; B = C$
  - Bryant reduction:
    - Replace $f(a)$ by A
    - Replace $f(b)$ by $ite(b = a, A, B)$
    - Replace $f(c)$ by $ite(c = a, A,\ ite(c = b, B, C)\ )$

Now, atoms are equalities between **constants**

Eager approach – Example (2)

- Second step: encode formula into propositional logic
  - Small-domain encoding:
    - If there are n different constants, there is a model with size at most n
    - $\log n$ bits to encode the value of each constant
    - $a = b$ translated using the bits for a and b
  - Per-constraint encoding:
    - Each atom $a = b$ is replaced by var $P_{a,b}$
    - Transitivity constraints are added (e.g. $P_{a,b} \wedge P_{b,c} \rightarrow P_{a,c}$)

This is a **very rough** overview of an encoding from EUF to SAT.

See [PRSS99, SSB02, SLB03, BGV01, BV02] for details.

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Lazy approach

Methodology:

Example: consider EUF and the CNF

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

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

Lazy approach

Methodology:

Example: consider EUF and the CNF

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

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

Lazy approach

Methodology:

Example: consider EUF and the CNF

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

- 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

Lazy approach

Methodology:

Example: consider EUF and the CNF

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

- 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 $[1, 2, 3, \overline{4}]$

Lazy approach

Methodology:

Example: consider EUF and the CNF

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

- 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

Lazy approach

Methodology:

Example: consider EUF and the CNF

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

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

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:
  Alt-Ergo, ArgoLib, Ario, Barcelogic, CVC, DTP, ICS, MathSAT, OpenSMT, Sateen, SVC, Simplify, tSAT, veriT, Yices, Z3, etc...

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Lazy approach - Optimizations

Several optimizations for enhancing efficiency:

- Check T-consistency only of full propositional models

Lazy approach - Optimizations

Several optimizations for enhancing efficiency:

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

Lazy approach - Optimizations

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

Lazy approach - Optimizations

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

Lazy approach - Optimizations

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

Lazy approach - Optimizations

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

Lazy approach - Important points

Important and benefitial aspects of the lazy approach:
(even with the optimizations)

- Everyone does what he/she is good at:
  - SAT solver takes care of Boolean information
  - Theory solver takes care of theory information
- Theory solver only receives conjunctions of literals
- Modular approach:
  - SAT solver and T-solver communicate via a simple API
  - SMT for a new theory only requires new T-solver
  - SAT solver can be embedded in a lazy SMT system with very few new lines of code

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Lazy approach - T-propagation

- As pointed out the lazy approach has one drawback:
  - Theory information does not guide the search (too lazy)
- How can we improve that?

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 basic lazy approach.
- Naive implementation::
  Add $\neg l$. If T-inconsistent then infer l [ACG00]
  But for efficient Theory Propagation we need:
  - -T-Solvers specialized and fast in it.
  - -fully exploited in conflict analysis
- This approach has been named DPLL(T) [NOT06]

DPLL(T)

In a nutshell:

$$\text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver}$$

- DPLL(X):
  - Very similar to a SAT solver, enumerates Boolean models
  - Not allowed: pure literal, blocked literal detection, ...
  - Required: incremental addition of clauses
  - Desirable: partial model detection
- T-Solver:
  - Checks consistency of conjunctions of literals
  - Computes theory propagations
  - Produces explanations of inconsistency/T-propagation
  - Should be incremental and backtrackable

DPLL(T) - Example

Consider again EUF and the formula:

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

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

DPLL(T) - Example

Consider again EUF and the formula:

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

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

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

DPLL(T) - Example

Consider again EUF and the formula:

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

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

DPLL(T) - Example

Consider again EUF and the formula:

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

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

DPLL(T) - Example

Consider again EUF and the formula:

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

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

DPLL(T) - Example

Consider again EUF and the formula:

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

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

UNSAT

DPLL(T) - Overall algorithm

High-levew view gives the same algorithm as a CDCL SAT solver:

```
while(true){

    while (propagate_gives_conflict()){
        if (decision_level==0) return UNSAT;
        else analyze_conflict();
    }

    restart_if_applicable();
    remove_lemmas_if_applicable();

    if (!decide()) returns SAT; // All vars assigned
}
```

Differences are in:

- `propagate_gives_conflict`
- `analyze_conflict`

DPLL(T) - Propagation

```
propagate_gives_conflict( ) returns Bool

    do {

      // unit propagate
      if ( unit_prop_gives_conflict() ) then return  true

      // check T-consistency of the model
      if ( solver.is_model_inconsistent() ) then return  true

      // theory propagate
      solver.theory_propagate()

    } while  (someTheoryPropagation)

    return false
```

DPLL(T) - Propagation (2)

- Three operations:
  - Unit propagation (SAT solver)
  - Consistency checks (T-solver)
  - Theory propagation (T-solver)
- Cheap operations are computed first
- If theory is expensive, calls to T-solver are sometimes skipped
- For completeness, only necessary to call T-solver at the leaves (i.e. when we have a full propositional model)
- Theory propagation is not necessary for completeness

Case Reasoning in Theory Solvers

- For certain theories, consistency checking requires case reasoning.

- Example: consider the theory of arrays and the set of literals

$$read(write(A,i,x),j) \neq x \qquad read(write(A,i,x),j) \neq read(A,j)$$

Case Reasoning in Theory Solvers

- For certain theories, consistency checking requires case reasoning.
- Example: consider the theory of arrays and the set of literals

  $$read(write(A,i,x),j) \neq x \qquad read(write(A,i,x),j) \neq read(A,j)$$

  Two cases:

  - $i = j$. LHS rewrites into $x \neq x$!!!

Case Reasoning in Theory Solvers

- For certain theories, consistency checking requires case reasoning.
- Example: consider the theory of arrays and the set of literals

  $$read(write(A,i,x),j) \neq x \qquad read(write(A,i,x),j) \neq read(A,j)$$

  Two cases:

  - $i = j$. LHS rewrites into $x \neq x$!!!
  - $i \neq j$. RHS rewrites into $read(A,j) \neq read(A,j)$!!!

Case Reasoning in Theory Solvers

- For certain theories, consistency checking requires case reasoning.
- Example: consider the theory of arrays and the set of literals

  $$read(write(A,i,x),j) \neq x \qquad read(write(A,i,x),j) \neq read(A,j)$$

  Two cases:

  - $i = j$. LHS rewrites into $x \neq x$!!!
  - $i \neq j$. RHS rewrites into $read(A,j) \neq read(A,j)$!!!

  CONCLUSION: T-inconsistent

Case Reasoning in Theory Solvers (2)

- A complete T-solver might need to reason by cases via internal case splitting and backtracking mechanisms.
- An alternative is to lift case splitting and backtracking from the T-Solver to the SAT engine.
- Basic idea: encode case splits as sets of clauses and send them as needed to the SAT engine for it to split on them.
- Possible benefits:
  - All case-splitting is coordinated by the SAT engine
  - Only have to implement case-splitting infrastructure in one place
  - Can learn a wider class of lemmas (more details later)

Case Reasoning in Theory Solvers (3)

- Basic idea: encode case splits as a set of clauses and send them as needed to the SAT engine
- Example:
  - Assume model contains literal $s = \underbrace{read(write(A, i, t), j)}_{s'}$
  - DPLL(X) asks: "is it T-satisfiable"?
  - T-solver says: "I do not know **yet**, but it will be helpful that you consider these theory lemmas:"

$$s = s' \wedge i = j \longrightarrow s = t$$

$$s = s' \wedge i \neq j \longrightarrow s = read(A, j)$$

- We need certain completeness conditions (e.g. once all lits from a certain subset $\mathcal{L}$ has been decided, the T-solver should answer YES/NO)

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

DPLL(T) - Conflict Analysis

Remember conflict analysis in SAT solvers:

C`:= conflicting clause`

while C `contains more than one lit of last DL`

  l`:=last literal assigned in` C
  C`:=Resolution(`C`,reason(`l`))`

end while

```
// let C = C' v l where l is UIP
backjump(maxDL(C'))
add l to the model with reason C
learn(C)
```

DPLL(T) - Conflict Analysis (2)

Conflict analysis in DPLL(T):

if `boolean conflict` **then** C`:= conflicting clause`
else C`:=`$\neg$`( solver.explain_inconsistency() )`

while C `contains more than one lit of last DL`

l`:=last literal assigned in` C
C`:=Resolution(`C`,reason(`l`))`

end while

```
// let C = C' v l where l is UIP
backjump(maxDL(C'))
add l to the model with reason C
learn(C)
```

DPLL(T) - Conflict Analysis (3)

What does `explain_inconsistency` return?

- A (small) conjuntion of literals $l_1 \wedge \ldots \wedge l_n$ such that:
  - They were in the model when T-inconsistency was found
  - It is T-inconsistent

What is now $reason(l)$?

- If l was unit propagated, reason is the clause that propagated it
- If l was T-propagated?
  - T-solver has to provide an explanation for l, i.e.
    a (small) set of literals $l_1, \ldots, l_n$ such that:
    - They were in the model when l was T-propagated
    - $l_1 \wedge \ldots \wedge l_n \models_T l$
  - Then $reason(l)$ is $\neg l_1 \vee \ldots \vee \neg l_n \vee l$

DPLL(T) - Conflict Analysis (4)

Let M be of the form $\ldots, c=b, \ldots$ and let F contain

$$h(a)=h(c) \;\vee\; p \qquad a=b \;\vee\; \neg p \;\vee\; a=d \qquad a\neq d \;\vee\; a=b$$

Take the following sequence:

1. Decide $h(a)\neq h(c)$
2. UnitPropagate p (due to clause $h(a)=h(c) \;\vee\; p$)
3. T-Propagate $a\neq b$ (since $h(a)\neq h(c)$ and $c=b$)
4. UnitPropagate $a=d$ (due to clause $a=b \;\vee\; \neg p \;\vee\; a=d$)
5. Conflicting clause $a\neq d \;\vee\; a=b$

Explain($a\neq b$) is $\{h(a)\neq h(c), c=b\}$

$$\dfrac{h(a)=h(c)\vee \mathbf{p} \qquad \dfrac{h(a)=h(c)\vee c\neq b\vee \mathbf{a\neq b} \qquad \dfrac{a=b\vee\neg p\vee \mathbf{a=d} \qquad \mathbf{a\neq d}\vee a=b}{\mathbf{a=b}\vee\neg p}}{h(a)=h(c)\vee c\neq b\vee\neg\mathbf{p}}}{h(a)=h(c) \;\vee\; c\neq b}$$

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - **Combining Theory Solvers**
  - Eager vs Lazy
  - Theory solver example

Need for combination

- In software verification, formulas like the following one arise:

$$a = b + 2 \wedge A = write(B, a+1, 4) \wedge (read(A, b+3) = 2 \vee f(a-1) \neq f(b+1))$$

- Here reasoning is needed over
  - The theory of linear arithmetic ($\mathbb{T}_{LA}$)
  - The theory of arrays ($\mathbb{T}_A$)
  - The theory of uninterpreted functions ($\mathbb{T}_{EUF}$)

- Remember that T-solvers only deal with conjunctions of lits.

- Given T-solvers for the three individual theories, can we combine them to obtain one for ($\mathbb{T}_{LA} \cup \mathbb{T}_A \cup \mathbb{T}_{EUF}$)?

- Under certain conditions the **Nelson-Oppen** combination method gives a positive answer

Motivating example - Convex case

Consider the following set of literals:

$$\begin{array}{rcl} f(f(x)-f(y)) & = & a \\ f(0) & = & a+2 \\ x & = & y \end{array}$$

There are two theories involved: $\mathbb{T}_{LA(\mathbb{R})}$ and $\mathbb{T}_{EUF}$

FIRST STEP: purify each literal so that it belongs to a single theory

$$f(f(x)-f(y)) = a \quad \Longrightarrow \quad \begin{array}{rcl} f(e_1) & = & a \\ e_1 & = & f(x)-f(y) \end{array} \quad \Longrightarrow \quad \begin{array}{rcl} f(e_1) & = & a \\ e_1 & = & e_2 - e_3 \\ e_2 & = & f(x) \\ e_3 & = & f(y) \end{array}$$

Motivating example - Convex case

Consider the following set of literals:

$$\begin{aligned} f(f(x)-f(y)) &= a \\ f(0) &= a+2 \\ x &= y \end{aligned}$$

There are two theories involved: $\mathbb{T}_{LA(\mathbb{R})}$ and $\mathbb{T}_{EUF}$

FIRST STEP: purify each literal so that it belongs to a single theory

$$f(0) = a+2 \quad \Longrightarrow \quad \begin{aligned} f(e_4) &= a+2 \\ e_4 &= 0 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} f(e_4) &= e_5 \\ e_4 &= 0 \\ e_5 &= a+2 \end{aligned}$$

Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

$$
\begin{array}{lcl}
 & EUF & \\
f(e_1) & = & a \\
f(x) & = & e_2 \\
f(y) & = & e_3 \\
f(e_4) & = & e_5 \\
x & = & y
\end{array}
\qquad\qquad
\begin{array}{lcr}
 & Arithmetic & \\
e_2 - e_3 & = & e_1 \\
e_4 & = & 0 \\
e_5 & = & a+2
\end{array}
$$

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

To merge the two models into a single one, the solvers have to agree on equalities between shared constants (interface equalities)

This can be done by exchanging entailed interface equalities

Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

$$
\begin{array}{rcl}
 & EUF & \\
f(e_1) & = & a \\
f(x) & = & e_2 \\
f(y) & = & e_3 \\
f(e_4) & = & e_5 \\
x & = & y
\end{array}
\qquad\qquad
\begin{array}{rcl}
 & Arithmetic & \\
e_2 - e_3 & = & e_1 \\
e_4 & = & 0 \\
e_5 & = & a+2 \\
e_2 & = & e_3
\end{array}
$$

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

- EUF-Solver says SAT
- Ari-Solver says SAT
- $EUF \models e_2 = e_3$

Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

EUF			$Arithmetic$		
$f(e_1)$	$=$	a	$e_2 - e_3$	$=$	e_1
$f(x)$	$=$	e_2	e_4	$=$	0
$f(y)$	$=$	e_3	e_5	$=$	$a+2$
$f(e_4)$	$=$	e_5	e_2	$=$	e_3
x	$=$	y			
e_1	$=$	e_4			

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

- EUF-Solver says SAT
- Ari-Solver says SAT
- $Ari \models e_1 = e_4$

Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

EUF			$Arithmetic$		
$f(e_1)$	$=$	a	$e_2 - e_3$	$=$	e_1
$f(x)$	$=$	e_2	e_4	$=$	0
$f(y)$	$=$	e_3	e_5	$=$	$a+2$
$f(e_4)$	$=$	e_5	e_2	$=$	e_3
x	$=$	y	a	$=$	e_5
e_1	$=$	e_4			

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

- EUF-Solver says SAT
- Ari-Solver says SAT
- $EUF \models a = e_5$

Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

$$
\begin{array}{ccc}
 & EUF & \\
f(e_1) & = & a \\
f(x) & = & e_2 \\
f(y) & = & e_3 \\
f(e_4) & = & e_5 \\
x & = & y \\
e_1 & = & e_4
\end{array}
\qquad
\begin{array}{ccc}
 & Arithmetic & \\
e_2 - e_3 & = & e_1 \\
e_4 & = & 0 \\
e_5 & = & a+2 \\
e_2 & = & e_3 \\
a & = & e_5
\end{array}
$$

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

- EUF-Solver says SAT
- Ari-Solver says **UNSAT**
- Hence the original set of lits was **UNSAT**

Nelson-Oppen – The convex case

- A theory T is stably-infinite iff every T-satisfiable quantifier-free formula has an infinite model
- A theory T is convex iff
  $S \models_T a_1 = b_1 \vee \ldots \vee a_n = b_n \implies S \models a_i = b_i$ for some i

Deterministic Nelson-Oppen: [NO79, TH96, MZ02]

- Given two signature-disjoint, stably-infinite and convex theories T_1 and T_2
- Given a set of literals S over the signature of $T_1 \cup T_2$
- The $(T_1 \cup T_2)$-satisfiability of S can be checked with the following algorithm:

Nelson-Oppen – The convex case (2)

Deterministic Nelson-Oppen

1. Purify S and split it into $S_1 \cup S_2$.
   Let $\mathcal{E}$ the set of interface equalities between S_1 and S_2
2. If S_1 is T_1-unsatisfiable then **UNSAT**
3. If S_2 is T_2-unsatisfiable then **UNSAT**
4. If $S_1 \models_{T_1} x = y$ with $x = y \in \mathcal{E} \setminus S_2$ **then**
   $S_2 := S_2 \cup \{x = y\}$ and **goto** 3
5. If $S_2 \models_{T_2} x = y$ with $x = y \in \mathcal{E} \setminus S_1$ **then**
   $S_1 := S_1 \cup \{x = y\}$ and **goto** 2
6. Report **SAT**

Motivating example – Non-convex case

Consider the following **UNSATISFIABLE** set of literals:

$$
\begin{array}{rcl}
1 \leq & x & \leq 2 \\
f(1) & = & a \\
f(x) & = & b \\
a & = & b + 2 \\
f(2) & = & f(1) + 3
\end{array}
$$

There are two theories involved: $\mathbb{T}_{LA(\mathbb{Z})}$ and $\mathbb{T}_{EUF}$

FIRST STEP: purify each literal so that it belongs to a single theory

$$
f(1) = a \quad \Longrightarrow \quad
\begin{array}{rcl}
f(e_1) & = & a \\
e_1 & = & 1
\end{array}
$$

Motivating example – Non-convex case

Consider the following **UNSATISFIABLE** set of literals:

$$\begin{array}{rcl} 1 \leq & x & \leq 2 \\ f(1) & = & a \\ f(x) & = & b \\ a & = & b+2 \\ f(2) & = & f(1)+3 \end{array}$$

There are two theories involved: $\mathbb{T}_{LA(\mathbb{Z})}$ and $\mathbb{T}_{EUF}$

FIRST STEP: purify each literal so that it belongs to a single theory

$$f(2) = f(1)+3 \implies \begin{array}{rcl} e_2 & = & 2 \\ f(e_2) & = & e_3 \\ f(e_1) & = & e_4 \\ e_3 & = & e_4+3 \end{array}$$

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

$$\begin{array}{ccc} & \mathit{Arithmetic} & \\ 1 & \leq & x \\ x & \leq & 2 \\ e_1 & = & 1 \\ a & = & b+2 \\ e_2 & = & 2 \\ e_3 & = & e_4+3 \\ a & = & e_4 \end{array} \qquad \begin{array}{ccc} & \mathit{EUF} & \\ f(e_1) & = & a \\ f(x) & = & b \\ f(e_2) & = & e_3 \\ f(e_1) & = & e_4 \end{array}$$

The two solvers only share constants: $x, e_1, a, b, e_2, e_3, e_4$

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- $EUF \models a = e_4$

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

Arithmetic	*EUF*
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b+2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4+3$	
$a = e_4$	

The two solvers only share constants: $x, e_1, a, b, e_2, e_3, e_4$

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- No theory entails any other interface equality, but...

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

Arithmetic	*EUF*
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b+2$	$f(e_1) = e_4$
$e_2 = 2$	
$e_3 = e_4 + 3$	
$a = e_4$	

The two solvers only share constants: $x, e_1, a, b, e_2, e_3, e_4$

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- $Ari \models_T x = e_1 \vee x = e_2$. Let's consider both cases.

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

Arithmetic	*EUF*
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b+2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_1$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_1$	

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- $EUF \models_T a = b$, that when sent to *Ari* makes it **UNSAT**

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

$$
\begin{array}{rcr}
\multicolumn{3}{c}{Arithmetic} \\
1 & \leq & x \\
x & \leq & 2 \\
e_1 & = & 1 \\
a & = & b+2 \\
e_2 & = & 2 \\
e_3 & = & e_4+3 \\
a & = & e_4
\end{array}
\qquad
\begin{array}{rcr}
\multicolumn{3}{c}{EUF} \\
f(e_1) & = & a \\
f(x) & = & b \\
f(e_2) & = & e_3 \\
f(e_1) & = & e_4
\end{array}
$$

Let's try now with $x = e_2$

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

Arithmetic	*EUF*
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_2$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_2$	

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- $EUF \models_T b = e_3$, that when sent to *Ari* makes it **UNSAT**

Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

Arithmetic	*EUF*
$1 \leq x$	$f(e_1) = a$
$x \leq 2$	$f(x) = b$
$e_1 = 1$	$f(e_2) = e_3$
$a = b + 2$	$f(e_1) = e_4$
$e_2 = 2$	$x = e_2$
$e_3 = e_4 + 3$	
$a = e_4$	
$x = e_2$	

Since both $x = e_1$ and $x = e_2$ are **UNSAT**, the set of literals is **UNSAT**

Nelson-Oppen - The non-convex case

- In the previous example Deterministic NO does not work
- This was because $T_{LA(\mathbb{Z})}$ is not convex:

  $S_{LA(\mathbb{Z})} \models_{T_{LA(\mathbb{Z})}} x = e_1 \vee x = e_2$, but

  $S_{LA(\mathbb{Z})} \not\models_{T_{LA(\mathbb{Z})}} x = e_1$ and

  $S_{LA(\mathbb{Z})} \not\models_{T_{LA(\mathbb{Z})}} x = e_2$
- However, there is a version of NO for non-convex theories
- Given a set constants $\mathcal{C}$, an arrangement $\mathcal{A}$ over $\mathcal{C}$ is:
  - A set of equalities and disequalites between constants in $\mathcal{C}$
  - For each $x, y \in \mathcal{C}$ either $x = y \in \mathcal{A}$ or $x \neq y \in \mathcal{A}$

Nelson-Oppen – The non-convex case (2)

Non-deterministic Nelson-Oppen: [NO79, TH96, MZ02]

- Given two signature-disjoint, stably-infinite theories T_1 and T_2
- Given a set of literals S over the signature of $T_1 \cup T_2$
- The $(T_1 \cup T_2)$-satisfiability of S can be checked via:

1. Purify S and split it into $S_1 \cup S_2$
   Let $\mathcal{C}$ be the set of shared constants
2. **For every** arrangement $\mathcal{A}$ over $\mathcal{C}$ **do**
   If $(S_1 \cup \mathcal{A})$ is T_1-satisfiable and $(S_2 \cup \mathcal{A})$ is T_2-satisfiable
   report **SAT**
3. Report **UNSAT**

This is another example of Case Reasoning inside a T-Solver

Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Eager vs Lazy Approach

REMEMBER....

Important and benefitial aspects of the lazy approach:
(even with the optimizations)

- Everyone does what he/she is good at:
  - SAT solver takes care of Boolean information
  - Theory solver takes care of theory information
- Theory solver only receives conjunctions of literals
- Modular approach:
  - SAT solver and T-solver communicate via a simple API
  - SMT for a new theory only requires new T-solver
  - SAT solver can be embedded in a lazy SMT system with very few new lines of code

Eager vs Lazy Approach (2)

- The Lazy Approach idea (*SAT Solver + Theory Reasoner*) has been applied to other extensions of SAT (x_i's are Boolean):
  - Cardinality constraints (e.g. $x_1 + x_2 + \ldots + x_7 \leq 4$)
  - Pseudo-Boolean constraints (e.g. $7x_1 + 4x_2 + 3x_3 + 5x_4 \leq 10$)
  - ...
- Also sophisticated encodings exist for these constraints (Eager Approach)
- Lazy approach extremely simple to implement, but is it always competitive w.r.t. an encoding?

Eager vs Lazy Approach (3)

Consider the problem with no SAT clauses and two constraints:

$$x_1 + \ldots + x_n \leq n/2$$
$$x_1 + \ldots + x_n > n/2$$

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:

Eager vs Lazy Approach (3)

Consider the problem with no SAT clauses and two constraints:

$$x_1 + \ldots + x_n \leq n/2$$
$$x_1 + \ldots + x_n > n/2$$

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:

$$\neg x_{i_1} \vee \ldots \vee \neg x_{i_{n/2+1}}$$
$$x_{i_1} \vee \ldots \vee x_{i_{n/2}}$$

Eager vs Lazy Approach (3)

Consider the problem with no SAT clauses and two constraints:

$$x_1 + \ldots + x_n \leq n/2$$
$$x_1 + \ldots + x_n > n/2$$

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:

$$\neg x_{i_1} \vee \ldots \vee \neg x_{i_{n/2+1}}$$
$$x_{i_1} \vee \ldots \vee x_{i_{n/2}}$$

- **All** $\binom{n}{\frac{n}{2}+1} + \binom{n}{n/2}$ explanations are needed to produce an unsatisfiable subset of clauses

Eager vs Lazy Approach (3)

Consider the problem with no SAT clauses and two constraints:

$$x_1 + \ldots + x_n \leq n/2$$
$$x_1 + \ldots + x_n > n/2$$

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:

$$\neg x_{i_1} \vee \ldots \vee \neg x_{i_{n/2+1}}$$
$$x_{i_1} \vee \ldots \vee x_{i_{n/2}}$$

- **All** $\binom{n}{\frac{n}{2}+1} + \binom{n}{n/2}$ explanations are needed to produce an unsatisfiable subset of clauses
- Hence, runtime is exponential in n.

Eager vs Lazy approach (4)

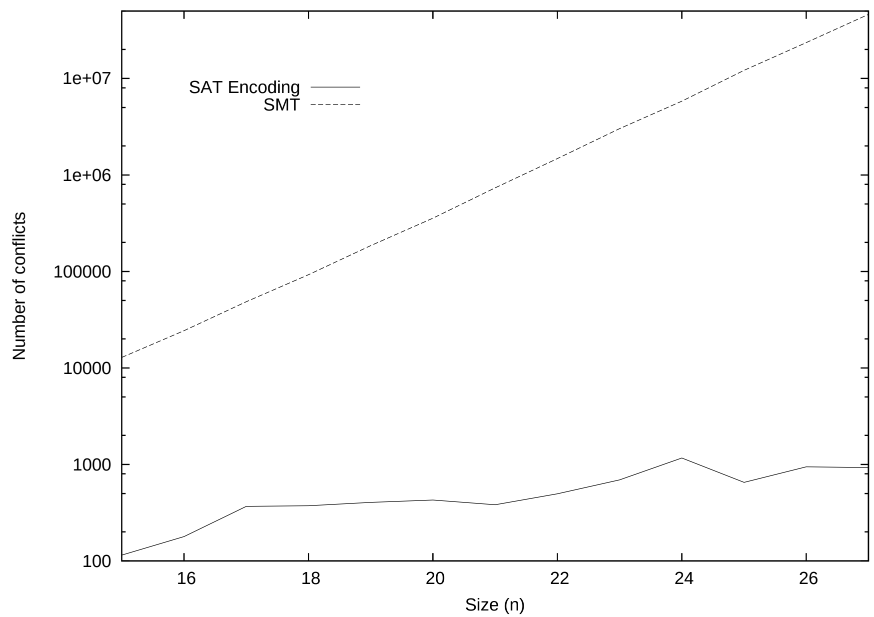
What has happened?

- Lazy approach = lazily encoding (parts of) the theory into SAT
- Sometimes, only parts of the theory need to be encoded
- But in this example the whole constraint is encoded into SAT...
- ...and the encoding used is a very naive one

Eager vs Lazy approach (4)

What has happened?

- Lazy approach = lazily encoding (parts of) the theory into SAT
- Sometimes, only parts of the theory need to be encoded
- But in this example the whole constraint is encoded into SAT...
- ...and the encoding used is a very naive one
- Best here is a good SAT encoding with auxiliary variables



Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
  - Theory solver example

Difference logic

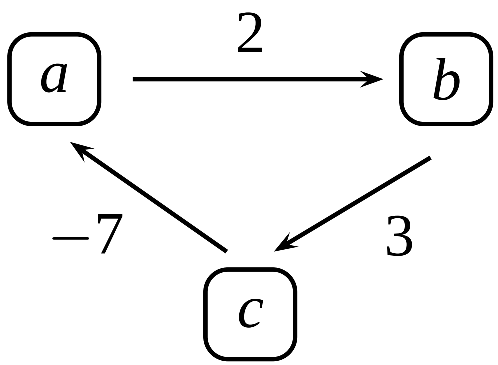
- Literals in Difference Logic are of the form $a - b \bowtie k$, where
  - $\bowtie \in \{\leq, \geq, <, >, =, \neq\}$
  - a and b are integer/real variables
  - k is an integer/real
- At the formula level,
  $a = b$ is replaced by p and $p \;\leftrightarrow\; a \leq b \wedge b \leq a$ is added
- If domain is $\mathbb{Z}$ then $a - b < k$ is replaced by $a - b \leq k - 1$
- If domain is $\mathbb{R}$ then $a - b < k$ is replaced by $a - b \leq k - \delta$
  - δ is a sufficiently small real
  - δ is not computed but used symbolically
    (i.e. numbers are pairs (k, δ))
- Hence we can assume all literals are $a - b \leq k$

Difference Logic - Remarks

- Note that any solution to a set of DL literals can be shifted
  (i.e. if σ is a solution then $\sigma'(x) = \sigma(x) + k$ also is a solution)
- This allows one to process bounds $x \leq k$
  - Introduce fresh variable *zero*
  - Convert all bounds $x \leq k$ into $x - zero \leq k$
  - Given a solution σ, shift it so that $\sigma(zero) = 0$
- If we allow (dis)equalities as literals, then:
  - If domain is $\mathbb{R}$ consistency check is polynomial
  - If domain is $\mathbb{Z}$ consistency check is NP-hard
    (k-colorability)
    - $1 \leq c_i \leq k$ with $i = 1 \ldots \#verts$ encodes k colors available
    - $c_i \neq c_j$ if i and j adjacents encode proper assignment

Difference Logic as a Graph Problem

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



- **Theorem:**

  M is T-inconsistent iff $\mathcal{G}(M)$ has a negative cycle

Difference Logic as a Graph Problem (2)

Theorem:

M is T-inconsistent iff $\mathcal{G}(M)$ has a negative cycle

$\Leftarrow$)

Any negative cycle $a_1 \xrightarrow{k_1} a_2 \xrightarrow{k_2} a_3 \longrightarrow \ldots \longrightarrow a_n \xrightarrow{k_n} a_1$ corresponds to a set of literals:

$$
\begin{array}{l}
a_1 - a_2 \leq k_1 \\
a_2 - a_3 \leq k_2 \\
\ldots \\
a_n - a_1 \leq k_n
\end{array}
$$

If we add them all, we get $0 \leq k_1 + k_2 + \ldots + k_n$, which is inconsistent since neg. cycle implies $k_1 + k_2 + \ldots + k_n < 0$

Difference Logic as a Graph Problem (3)

Theorem:

$$M \text{ is } T\text{-inconsistent iff } \mathcal{G}(M) \text{ has a negative cycle}$$

$\Rightarrow)$
Let us assume that there is no negative cycle.

1. Consider additional vertex o with edges $o \xrightarrow{0} v$ for all verts. v
2. For each variable x, let $\sigma(x) = -dist(o,x)$
3. σ is a model of M
   - If $\sigma \not\models x - y \leq k$ then $-dist(o,x) + dist(o,y) > k$
   - Hence, $dist(o,y) > dist(o,x) + k$
   - But $k = weight(x \longrightarrow y)$!!!

Difference Logic as a Graph Problem (3)

Theorem:

M is T-inconsistent iff $\mathcal{G}(M)$ has a negative cycle

$\Rightarrow)$
Let us assume that there is no negative cycle.

1. Consider additional vertex o with edges $o \xrightarrow{0} v$ for all verts. v
2. For each variable x, let $\sigma(x) = -dist(o,x)$
3. σ is a model of M
   - If $\sigma \not\models x - y \leq k$ then $-dist(o,x) + dist(o,y) > k$
   - Hence, $dist(o,y) > dist(o,x) + k$
   - But $k = weight(x \longrightarrow y)$!!!

Where am I using there is no negative cycle?

Difference Logic as a Graph Problem (3)

Theorem:

M is T-inconsistent iff $\mathcal{G}(M)$ has a negative cycle

$\Rightarrow$)
Let us assume that there is no negative cycle.

1. Consider additional vertex o with edges $o \xrightarrow{0} v$ for all verts. v
2. For each variable x, let $\sigma(x) = -dist(o,x)$
   [exists because there is no negative cycle]
3. σ is a model of M
   - If $\sigma \not\models x - y \leq k$ then $-dist(o,x) + dist(o,y) > k$
   - Hence, $dist(o,y) > dist(o,x) + k$
   - But $k = weight(x \longrightarrow y)$!!!

Where am I using there is no negative cycle?

Bellman-Ford: negative cycle detection

forall $v \in V$ **do** $d[v] := \infty$ **endfor**
$d[origin] = 0$
forall $i = 1$ **to** $|V| - 1$ **do**
    forall $(u,v) \in E$ **do**
        if `d[v] >  d[u] + weight(u,v)` **then**
            `d[v]:= d[u] + weight(u,v)`
            `p[v]:= u`
        endif
    endfor
endfor

forall $(u,v) \in E$ **do**
    if $d[v] > d[u] + weight(u,v)$ **then**
        `Negative cycle detected`
        `Cycle reconstructed following` p
    endif
endfor

Consistency checks

- Consistency checks can be performed using Bellman-Ford in time ($O(|V| \cdot |E|)$)
- Other more efficient variants exists[WIGG05, SM06].
- Incrementality easy:
  - Upon arrival of new literal $a \xrightarrow{k} b$ process graph from a
- Solutions can be kept after backtracking
- Inconsistency explanations are negative cycles
  (irredundant but not minimal explanations)

Theory propagation

- Addition of $a \xrightarrow{k} b$ entails $c - d \leq k'$ only if

$$\rlap{$\underbrace{\phantom{c \longrightarrow^{*} a \xrightarrow{k} b}}_{\mathit{shortest}}$}c \longrightarrow^{*} \overbrace{a \xrightarrow{k} b \longrightarrow^{*} d}^{\mathit{shortest}}$$

- Each edge $a \xrightarrow{k} b$ has its reduced cost $-\sigma(a) + \sigma(b) + k \geq 0$
- Shortest path computation more efficient using reduced costs, since they are non-negative [Dijkstra's algorithm]
- Theory propagation $\approx$ shortest-path computations
- Explanations are the shortest paths

Bibliography - Some further reading

- Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli. *Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T).* J. ACM 53(6): 937-977 (2006)
- Roberto Sebastiani. *Lazy Satisfiability Modulo Theories*. JSAT 3(3-4): 141-224 (2007).
- Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli. *Satisfiability Modulo Theories*. Handbook of Satisfiability 2009: 825-885

References

[ABC+02] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT-Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In A. Voronkov, editor, *18th International Conference on Automated Deduction, CADE'02*, volume 2392 of *Lecture Notes in Conference Science*, pages 195–210. Springer, 2002.

[ACG00] A. Armando, C. Castellini, and E. Giunchiglia. SAT-Based Procedures for Temporal Reasoning. In S. Biundo and M. Fox, editors, *5th European Conference on Planning, ECP'99*, volume 1809 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2000.

[AMP06] A. Armando, J. Mantovani, and L. Platania. Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers. In A. Valmari, editor, *13th International SPIN Workshop, SPIN'06*, volume 3925 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2006.

[BB09] R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In S. Kowalewski and A. Philippou, editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'05*, volume 5505 of *Lecture Notes in Computer Science*, pages 174–177. Springer, 2009.

References

[BBC$^+$05] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'05*, volume 3576 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2005.

[BCF$^+$07] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A Lazy and Layered SMT(BV) Solver for Hard Industrial Verification Problems. In W. Damm and H. Hermanns, editors, *19th International Conference on Computer Aided Verification, CAV'07*, volume 4590 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2007.

[BD94] J. R. Burch and D. L. Dill. Automatic Verification of Pipelined Microprocessor Control. In D. L. Dill, editor, *6th International Conference on Computer Aided Verification, CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 68–80. Springer, 1994.

[BDS02a] C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation into SAT. In E. Brinksma and K. G. Larsen, editors, *14th International Conference on Computer Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2002.

References

[BDS02b] C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation into SAT. In E. Brinksma and K. G. Larsen, editors, *14th International Conference on Computer Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2002.

[BGV01] R. E. Bryant, S. M. German, and M. N. Velev. Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic. *ACM Transactions on Computational Logic, TOCL*, 2(1):93–134, 2001.

[BLNM+09] C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic. In R. A. Schmidt, editor, *22nd International Conference on Automated Deduction , CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2009.

[BM90] R. S. Boyer and J. S. Moore. A Theorem Prover for a Computational Logic. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction, CADE'90*, volume 449 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1990.

[BNO+08a] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. A Write-Based Solver for SAT Modulo the Theory of Arrays. In *Formal Methods in Computer-Aided Design, FMCAD*, pages 1–8, 2008.

References

[BNO$^+$08b] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. The barcelogic smt solver. In *Computer-aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298, 2008.

[BV02] R. E. Bryant and M. N. Velev. Boolean Satisfiability with Transitivity Constraints. *ACM Transactions on Computational Logic, TOCL*, 3(4):604–627, 2002.

[DdM06] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In T. Ball and R. B. Jones, editors, *18th International Conference on Computer Aided Verification, CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[dMB09] L. de Moura and N. Bjørner. Generalized, efficient array decision procedures. In *9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009*, pages 45–52. IEEE, 2009.

[dMR02] L. de Moura and H. Rueß. Lemmas on Demand for Satisfiability Solvers. In *5th International Conference on Theory and Applications of Satisfiability Testing, SAT'02*, pages 244–251, 2002.

[DNS05] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM, JACM*, 52(3):365–473, 2005.

References

[FORS01] J. Filliâtre, S. Owre, H. Rueß, and Natarajan Shankar. ICS: Integrated Canonization and Solving (Tool presentation). In G. Berry, H. Comon, and A. Finkel, editors, *13th International Conference on Computer Aided Verification, CAV'01*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer, 2001.

[LM05] S. K. Lahiri and M. Musuvathi. An Efficient Decision Procedure for UTVPI Constraints. In B. Gramlich, editor, *5th International Workshop on Frontiers of Combining Systems, FroCos'05*, volume 3717 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2005.

[LNO06] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT Techniques for Fast Predicate Abstraction. In T. Ball and R. B. Jones, editors, *18th International Conference on Computer Aided Verification, CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 413–426. Springer, 2006.

[LS04] S. K. Lahiri and S. A. Seshia. The UCLID Decision Procedure. In R. Alur and D. Peled, editors, *16th International Conference on Computer Aided Verification, CAV'04*, volume 3114 of *Lecture Notes in Computer Science*, pages 475–478. Springer, 2004.

[MZ02] Z. Manna and C. G. Zarba. Combining Decision Procedures. In B. K. Aichernig and T. S. E. Maibaum, editors, *10th Anniversary Colloquium of UNU/IIST*, volume 2757 of *Lecture Notes in Computer Science*, pages 381–422. Springer, 2002.

References

[NO80] G. Nelson and D. C. Oppen. Fast Decision Procedures Based on Congruence Closure. *Journal of the ACM, JACM*, 27(2):356–364, 1980.

[NO05] R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'05*, volume 3576 of *Lecture Notes in Computer Science*, pages 321–334. Springer, 2005.

[NO07] R. Nieuwenhuis and A. Oliveras. Fast Congruence Closure and Extensions. *Information and Computation, IC*, 2005(4):557–580, 2007.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding Equality Formulas by Small Domains Instantiations. In N. Halbwachs and D. Peled, editors, *11th International Conference on Computer Aided Verification, CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1999.

References

[SBDL01] A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *16th Annual IEEE Symposium on Logic in Computer Science, LICS'01*, pages 29–37. IEEE Computer Society, 2001.

[Sha02] N. Shankar. Little Engines of Proof. In L. H. Eriksson and P. A. Lindsay, editors, *International Symposium of Formal Methods Europe, FME'02*, volume 2391 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2002.

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.

[SLB03] S. Seshia, S. K. Lahiri, and R. E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic wit h Uninterpreted Functions. In *40th Design Automation Conference, DAC'03*, pages 425–430. ACM Press, 2003.

[SM06] S.Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2006.

[SSB02] O. Strichman, S. A. Seshia, and R. E. Bryant. Deciding Separation Formulas with SAT. In E. Brinksma and K. G. Larsen, editors, *14th International Conference on Computer Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2002.

References

[TdH08] N. Tillmann and J. de Halleux. Pex-White Box Test Generation for .NET. In B. Beckert and R. Hähnle, editors, *2nd International Conference on Tests and Proofs, TAP'08*, volume 4966 of *Lecture Notes in Computer Science*, pages 134–153. Springer, 2008.

[TH96] C. Tinelli and M. T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In *Procs. Frontiers of Combining Systems (FroCoS)*, Applied Logic, pages 103–120. Kluwer Academic Publishers, March 1996.

[WIGG05] C. Wang, F. Ivancic, M. K. Ganai, and A. Gupta. Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination. In G. Sutcliffe and A. Voronkov, editors, *12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2005.

[ZM10] H. Zankl and A. Middeldorp. Satisfiability of Non-linear (Ir)rational Arithmetic. In Edmund M. Clarke and Andrei Voronkov, editors, *16th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'10*, volume 6355 of *Lecture Notes in Computer Science*, pages 481–500. Springer, 2010.