2. SMT Solving: Eager vs Lazy Approaches

Prof. Roberto Amadini

Department of Computer Science and Engineering, University of Bologna, Italy

Combinatorial Decision Making and Optimization

2$^{\text{nd}}$ cycle degree programme in Artificial Intelligence

University of Bologna, Academic Year 2023/24

SMT solving

- SMT is an extension of SAT
- Unsurprisingly, SMT solving relies on SAT solving
  - SMT solving $\equiv$ finding (if any) a $\mathcal{T}$-model satisfying a $\mathcal{T}$-formula φ
- How to encode SMT formulas to corresponding SAT formulas?
  - Eager approaches
  - Lazy approaches
  - *Hybrid* approaches

Eager approaches

- Eager approaches translate upfront SMT formulas to equisatisfiable SAT formulas
  - φ, φ' are equisatisfiable iff φ has a model $\mathcal{M} \iff \varphi'$ has *a* model $\mathcal{M}'$
  - all theory information is used from the beginning
- Eager encodings are naturally theory-specific
- Pros:
  - No need of theory solvers
  - once SMT formula encoded, we can use best available SAT solver(s)
- Cons:
  - Complex, *ad hoc* encodings needed for all the theories we use
  - Resulting SAT formula can be huge

Eager approaches

- E.g., consider a EUF formula φ. Instead of looking for a $\mathcal{T}_{EUF}$-model, we encode it into an equisatisfiable SAT formula φ^p:
- First step: replace function/predicate with constant equalities
- E.g., suppose we have terms $f(a)$, $f(b)$, $f(c)$:
- Ackermann approach: replace $f(a), f(b), f(c)$ with new constants A, B, C and add $a = b \to A = B, a = c \to A = C, b = c \to B = C$
- Bryant approach:
  - replace $f(a)$ by A
  - replace $f(b)$ by $ite(a = b, A, B)$
  - replace $f(c)$ by $ite(a = c, A, ite(b = c, B, C))$

Example

- E.g., suppose we have $p(x, y, y)$ and $p(x, z, t)$. We add P_1, P_2 and:
- Ackermann: replace $p(x, y, y), p(x, z, t)$ with P_1, P_2 and add formula $(x = x \wedge y = z \wedge y = t) \rightarrow P_1 = P_2$
  - i.e., $y \neq z \vee y \neq t \vee P_1 = P_2$
- Bryant: replace $p(x, y, y)$ with P_1 and $p(x, z, t)$ with $ite(x = x \wedge y = z \wedge y = t, P_1, P_2)$

SAT Encodings

- Second step: remove equalities to reduce φ in propositional logic
- Small-domain encoding: if φ has n distinct uninterpreted constants $\{c_1, \ldots, c_n\}$, a model $\mathcal{M} = \langle M, (\cdot)^{\mathcal{M}} \rangle$ for φ has size $|M| \leq n$
  - we don't have functions/predicates anymore, only equalities
- Each $c_i^{\mathcal{M}}$ can be interpreted in $\{1, \ldots, n\}$:
  - We only care if $c_i = c_j$ or $c_i \neq c_j$, we don't care about $|c_i - c_j|$
  - Each $c_i^{\mathcal{M}}$ takes $O(\log n)$ bits $\rightarrow$ overall $O(n \log n)$ space complexity
  - $a = b$ encoded to SAT using the bits for a and b
- Direct encoding (a.k.a. per-constraint encoding):
  - Replace each $a = b$ with a propositional symbol $P_{a,b}$
  - Add transitivity constraints of the form $(P_{a,b} \wedge P_{b,c}) \rightarrow P_{a,c}$

Which encoding?

- Small-domain and direct encoding are different ways of translating SMT$\rightarrow$ SAT. *Which one should be used?*
- No general answer: it depends on the problem structure
  - Algorithm selection (AS) problem
- Direct encoding may generate larger problems solved quickly
  - Also depending on the underlying SAT solver(s)
- AS techniques enable to choose/combine different encodings
  - The first ML-based approach dates back to 2005 (it used SVMs): *Sanjit A. Seshia. Adaptive Eager Boolean Encoding for Arithmetic Reasoning in Verification. PhD thesis, Carnegie Mellon University*

Lazy approaches

- **Lazy approach**: instead of compiling SMT problems to SAT, we integrate SAT solvers with theory-specific decision procedures

- Most SMT solvers are nowadays lazy: SAT solvers + theory-specific solvers ($\mathcal{T}$-solvers)
  - Theory information used lazily, when checking $\mathcal{T}$-consistency of the Boolean abstraction for the input $\mathcal{T}$-formula

- A $\mathcal{T}$-solver takes in input a conjunction of $\mathcal{T}$-literals φ and decides whether φ is satisfiable
  - i.e., wheter it exists a $\mathcal{T}$-model $\mathcal{M}$ s.t. $\varphi^{\mathcal{M}} = \mathit{true}$

- Pros: more modular and flexible, no blow-up of SAT clauses
- Cons: search is SAT-driven rather than $\mathcal{T}$-driven

Example

- Consider e.g. the EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg\ell_4}$$

BOOLEAN ABSTRACTION

- φ abstracted into SAT formula $\ell_1 \wedge (\neg\ell_2 \vee \ell_3) \wedge \neg\ell_4$ in CNF
  - Also written $\Phi = \{\ell_1, \neg\ell_2 \vee \ell_3, \neg\ell_4\}$

Example

- Consider e.g. the EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg \ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg \ell_4}$$

- φ abstracted into SAT formula $\ell_1 \wedge (\neg \ell_2 \vee \ell_3) \wedge \neg \ell_4$ in CNF
  - Also written $\Phi = \{\ell_1, \neg \ell_2 \vee \ell_3, \neg \ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver sends Φ to SAT solver, returning model $\mathcal{M} = \{\ell_1, \neg \ell_2, \neg \ell_4\}$

Example

- Consider e.g. the EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg\ell_4}$$

- φ abstracted into SAT formula $\ell_1 \wedge (\neg\ell_2 \vee \ell_3) \wedge \neg\ell_4$ in CNF
  - Also written $\Phi = \{\ell_1, \neg\ell_2 \vee \ell_3, \neg\ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver sends Φ to SAT solver, returning model $\mathcal{M} = \{\ell_1, \neg\ell_2, \neg\ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver says $\mathcal{M}$ is $\mathcal{T}$-inconsistent and sends back to SAT solver formula $\Phi' = \Phi \cup \neg\mathcal{M} = \{\ell_1, \neg\ell_2 \vee \ell_3, \neg\ell_4, \neg\ell_1 \vee \ell_2 \vee \ell_4\}$

Example

- Consider e.g. the EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \ \wedge \ (\underbrace{f(g(a)) \neq f(c)}_{\neg \ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \ \wedge \ \underbrace{c \neq d}_{\neg \ell_4}$$

- φ abstracted into SAT formula $\ell_1 \wedge (\neg \ell_2 \vee \ell_3) \wedge \neg \ell_4$ in CNF
    - Also written $\Phi = \{\ell_1, \neg \ell_2 \vee \ell_3, \neg \ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver sends Φ to SAT solver, returning model $\mathcal{M} = \{\ell_1, \neg \ell_2, \neg \ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver says $\mathcal{M}$ is $\mathcal{T}$-inconsistent and sends back to SAT solver formula $\Phi' = \Phi \cup \neg \mathcal{M} = \{\ell_1, \neg \ell_2 \vee \ell_3, \neg \ell_4, \neg \ell_1 \vee \ell_2 \vee \ell_4\}$
- SAT solver returns model $\mathcal{M}' = \{\ell_1, \ell_2, \ell_3, \neg \ell_4\}$

Example

- Consider e.g. the EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg\ell_4}$$

- φ abstracted into SAT formula $\ell_1 \wedge (\neg\ell_2 \vee \ell_3) \wedge \neg\ell_4$ in CNF
  - Also written $\Phi = \{\ell_1, \neg\ell_2 \vee \ell_3, \neg\ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver sends Φ to SAT solver, returning model $\mathcal{M} = \{\ell_1, \neg\ell_2, \neg\ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver says $\mathcal{M}$ is $\mathcal{T}$-inconsistent and sends back to SAT solver formula $\Phi' = \Phi \cup \neg\mathcal{M} = \{\ell_1, \neg\ell_2 \vee \ell_3, \neg\ell_4, \neg\ell_1 \vee \ell_2 \vee \ell_4\}$
- SAT solver returns model $\mathcal{M}' = \{\ell_1, \ell_2, \ell_3, \neg\ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver says $\mathcal{M}'$ is $\mathcal{T}$-inconsistent and sends back $\Phi'' = \Phi' \cup \neg\mathcal{M}' = \{\ell_1, \neg\ell_2 \vee \ell_3, \neg\ell_4, \neg\ell_1 \vee \ell_2 \vee \ell_4, \neg\ell_1 \vee \neg\ell_2 \vee \neg\ell_3 \vee \ell_4\}$

Example

- Consider e.g. the EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \ \wedge \ (\underbrace{f(g(a)) \neq f(c)}_{\neg \ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \ \wedge \ \underbrace{c \neq d}_{\neg \ell_4}$$

- φ abstracted into SAT formula $\ell_1 \wedge (\neg \ell_2 \vee \ell_3) \wedge \neg \ell_4$ in CNF
  - Also written $\Phi = \{\ell_1, \neg \ell_2 \vee \ell_3, \neg \ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver sends Φ to SAT solver, returning model $\mathcal{M} = \{\ell_1, \neg \ell_2, \neg \ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver says $\mathcal{M}$ is $\mathcal{T}$-inconsistent and sends back to SAT solver formula $\Phi' = \Phi \cup \neg \mathcal{M} = \{\ell_1, \neg \ell_2 \vee \ell_3, \neg \ell_4, \neg \ell_1 \vee \ell_2 \vee \ell_4\}$
- SAT solver returns model $\mathcal{M}' = \{\ell_1, \ell_2, \ell_3, \neg \ell_4\}$
- $\mathcal{T}_{\mathcal{E}}$-solver says $\mathcal{M}'$ is $\mathcal{T}$-inconsistent and sends back $\Phi'' = \Phi' \cup \neg \mathcal{M}' = \{\ell_1, \neg \ell_2 \vee \ell_3, \neg \ell_4, \neg \ell_1 \vee \ell_2 \vee \ell_4, \neg \ell_1 \vee \neg \ell_2 \vee \neg \ell_3 \vee \ell_4\}$
- SAT solver detects Φ'' unsatisfiable

Example

$$\Phi'' \equiv \ell_1 \wedge (\neg \ell_2 \vee \ell_3) \vee \neg \ell_4 \vee (\neg \ell_1 \vee \ell_2 \vee \ell_4) \vee (\neg \ell_1 \vee \neg \ell_2 \vee \neg \ell_3 \vee \ell_4)$$

ℓ_1	ℓ_2	ℓ_3	ℓ_4	Φ''
true	*true*	*true*	*true*	*false*
true	*true*	*true*	*false*	*false*
true	*true*	*false*	*true*	*false*
true	*true*	*false*	*false*	*false*
true	*false*	*true*	*true*	*false*
true	*false*	*true*	*false*	*false*
true	*false*	*false*	*true*	*false*
true	*false*	*false*	*false*	*false*
false	*true*	*true*	*true*	*false*
false	*true*	*true*	*false*	*false*
false	*true*	*false*	*true*	*false*
false	*true*	*false*	*false*	*false*
false	*false*	*true*	*true*	*false*
false	*false*	*true*	*false*	*false*
false	*false*	*false*	*true*	*false*
false	*false*	*false*	*false*	*false*

Basic idea

Require: φ is a qff in the signature Σ of T
Ensure: output is sat if φ is T-satisfiable, and unsat otherwise

```
F := φ^a            abstract the formula
loop
  A := get_model(F)     try to get a solution (SAT)
  if A = none then
    return unsat        CONJUNCTION OF LITERALS OF A
  else
    μ := check_sat_T(A^c)
    if μ = sat then
      return sat
    else
      F := F ∧ ¬μ^a     Fix the boolean abstraction with the negation of
                        the model
```

Fig. 1 A basic SMT solver based on the lazy approach. The function get_model implements the SAT engine. It takes a propositional formula F and returns either none, if F is unsatisfiable, or a satisfiable conjunction A of propositional literals such that $A \models F$. The function $\mathsf{check\_sat}_T$ implements the theory solver. It takes a conjunction ψ of Σ-literals and returns either sat or a T-unsatisfiable conjunction μ of literals from ψ.

Figure from: Barrett, Clark, and Cesare Tinelli. "Satisfiability modulo theories." Handbook of model checking. Springer, Cham, 2018. 305-343.

Lazy approaches

Lazy approaches can implement several performance optimizations:

- $\mathcal{T}$-consistency can be checked dynamically on partial assignments
  - instead of always considering the full propositional model
- Given $\mathcal{T}$-inconsistent assignment μ identify a smaller, $\mathcal{T}$-inconsistent $\eta \subseteq \mu$ and return $\neg\eta$
  - instead of always returning $\neg\mu$
- Upon $\mathcal{T}$-inconsistency, backjump to a $\mathcal{T}$-consistent point in the computation
  - instead of systematic chronological backtracking

Lazy approaches

- Lazy approaches have important benefits w.r.t. eager approaches:
- Everyone does what is good at:
  - SAT solvers take care of Boolean information
    - SAT clauses (in CNF) of the Boolean abstraction
  - Theory solvers take care of theory information
    - only conjunctions of literals, corresponding to (partial) assignments
- Modular approach:
  - SAT/SMT solvers communicate via simple APIs
  - SAT solvers can be embedded in lazy SMT solvers with little effort
  - Adding a new theory $\mathcal{T}$ only requires plugging in a new $\mathcal{T}$-solver

CDCL($\mathcal{T}$)

- In a nutshell, CDCL($\mathcal{T}$) $\simeq$ CDCL $+$ $\mathcal{T}$-solver
  - CDCL approach to SAT solving is extended to enumerate truth values whose $\mathcal{T}$-satisfiability is checked by a $\mathcal{T}$-solver
- $\mathcal{T}$-solver:
  - Checks consistency of conjunctions of literals
  - Possibly performs deductions of unassigned literals ($\mathcal{T}$-propagation)
  - Produces explanations of inconsistent assignments
  - Should be incremental and backtrackable

Abstract framework

- Let's see how the above example is decided via an abstract framework based on state transitions of the form $\mu \parallel \varphi \implies \mu' \parallel \varphi'$ s.t.
  - φ, φ' are $\mathcal{T}$-formulas
  - μ, μ' are (partial) Boolean assignments to atoms of φ, φ' resp.
  - $\mu \parallel \varphi$ and $\mu' \parallel \varphi'$ are called states
  - Each transition $\mu \parallel \varphi \implies \mu' \parallel \varphi'$ is defined by transition rules
  - A sequence of transitions is called derivation
- If from initial state $\emptyset \parallel \varphi$ we soundly derive a final state $\mu \parallel \varphi$ where μ is a complete assignment of φ, then φ is $\mathcal{T}$-consistent ($\mu \models_{\mathcal{T}} \varphi$)

Why an abstract framework?

- Skip over implementation details and unimportant control aspects
- Reason formally about solvers for SAT and SMT
- Model advanced features such as non-chronological backtracking, lemma learning, theory propagation, ...
- Describe different strategies and prove their correctness
- Compare different systems at a higher level

CDCL($\mathcal{T}$) example

- Consider again EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg\ell_4}$$

- Initial state: $\emptyset \parallel \varphi$

CDCL($\mathcal{T}$) example

- Consider again EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg\ell_4}$$

- Initial state: $\emptyset \parallel \varphi$
- Unit propagate rule: $\emptyset \parallel \varphi \implies \{\ell_1\} \parallel \varphi$

CDCL($\mathcal{T}$) example

- Consider again EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg \ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg \ell_4}$$

- Initial state: $\emptyset \parallel \varphi$
- Unit propagate rule: $\emptyset \parallel \varphi \implies \{\ell_1\} \parallel \varphi$
- Unit propagate rule: $\{\ell_1\} \parallel \varphi \implies \{\ell_1, \neg \ell_4\} \parallel \varphi$

CDCL($\mathcal{T}$) example

- Consider again EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg\ell_4}$$

- Initial state: $\emptyset \parallel \varphi$
- Unit propagate rule: $\emptyset \parallel \varphi \implies \{\ell_1\} \parallel \varphi$
- Unit propagate rule: $\{\ell_1\} \parallel \varphi \implies \{\ell_1, \neg\ell_4\} \parallel \varphi$
- $\mathcal{T}$-propagate rule: $\{\ell_1, \neg\ell_4\} \parallel \varphi \implies \{\ell_1, \neg\ell_4, \ell_2\} \parallel \varphi$

CDCL($\mathcal{T}$) example

- Consider again EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \;\wedge\; (\underbrace{f(g(a)) \neq f(c)}_{\neg \ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \;\wedge\; \underbrace{c \neq d}_{\neg \ell_4}$$

- Initial state: $\emptyset \parallel \varphi$
- Unit propagate rule: $\emptyset \parallel \varphi \implies \{\ell_1\} \parallel \varphi$
- Unit propagate rule: $\{\ell_1\} \parallel \varphi \implies \{\ell_1, \neg \ell_4\} \parallel \varphi$
- $\mathcal{T}$-propagate rule: $\{\ell_1, \neg \ell_4\} \parallel \varphi \implies \{\ell_1, \neg \ell_4, \ell_2\} \parallel \varphi$
- $\mathcal{T}$-propagate rule: $\{\ell_1, \neg \ell_4, \ell_2\} \parallel \varphi \implies \{\ell_1, \neg \ell_4, \ell_2, \neg \ell_3\} \parallel \varphi$

CDCL($\mathcal{T}$) example

- Consider again EUF formula φ:

$$\underbrace{g(a) = c}_{\ell_1} \ \wedge\ (\underbrace{f(g(a)) \neq f(c)}_{\neg\ell_2} \vee \underbrace{g(a) = d}_{\ell_3}) \ \wedge\ \underbrace{c \neq d}_{\neg\ell_4}$$

- Initial state: $\emptyset \parallel \varphi$
- Unit propagate rule: $\emptyset \parallel \varphi \implies \{\ell_1\} \parallel \varphi$
- Unit propagate rule: $\{\ell_1\} \parallel \varphi \implies \{\ell_1, \neg\ell_4\} \parallel \varphi$
- $\mathcal{T}$-propagate rule: $\{\ell_1, \neg\ell_4\} \parallel \varphi \implies \{\ell_1, \neg\ell_4, \ell_2\} \parallel \varphi$
- $\mathcal{T}$-propagate rule: $\{\ell_1, \neg\ell_4, \ell_2\} \parallel \varphi \implies \{\ell_1, \neg\ell_4, \ell_2, \neg\ell_3\} \parallel \varphi$
- Fail rule: $\{\ell_1, \neg\ell_4, \ell_2, \neg\ell_3\} \parallel \varphi \implies$ *Fail*
  - We are at decision level 0 (no literal decided), so φ unsatisfiable

$\mathcal{T}$-propagation

- $\mathcal{T}$-propagation makes lazy approaches "less lazy": theory information can guide the search via deductions or $\mathcal{T}$-consequences
- General rule:
  - if $\mu \models_{\mathcal{T}} \ell$, *and*
  - ℓ or $\neg\ell$ occurs in φ, *and*
  - neither ℓ nor $\neg\ell$ occur in μ, *then*:

  $$\mu \parallel \varphi \implies \mu \cup \{\ell\} \parallel \varphi$$

- E.g., if $\mathcal{T} = \mathcal{T}_{\mathcal{Z}}$ and $a < b, b < c \in \mu$ then $\mu \models_{\mathcal{T}} (a < c)$
- If neither $a < c$ nor $\neg(a < c) \equiv a \geq c$ occur in μ, and φ contains $a < c$ or $a \geq c$, then we can add $a < c$ to μ

CDCL($\mathcal{T}$) algorithm

function $\mathcal{T}$-CDCL(φ : $\mathcal{T}$-formula, μ : $\mathcal{T}$-assignment)
    if *preProcess*$(\varphi, \mu) =$ *Conflict* **then return** $\perp$ ▷ Pre-processing
    $\varphi^p \leftarrow \mathcal{T}2\mathcal{B}(\varphi)$; $\mu^p \leftarrow \mathcal{T}2\mathcal{B}(\mu)$ ▷ Boolean abstractions
    level $\leftarrow 0$ ▷ Decision level 0
    while *true* **do**
        status $\leftarrow$ *propagate*(φ^p, μ^p, ℓ) ▷ Unit/$\mathcal{T}$-propagation
        if *status* $=$ SAT **then return** $\mathcal{B}2\mathcal{T}(\mu^p)$ ▷ φ satisfiable
        else if *status* $=$ UNSAT **then**
            level $\leftarrow$ *analyzeConflict*(φ^p, μ^p) ▷ Conflict analysis
            if *level* < 0 **then return** $\perp$ ▷ φ unsatisfiable
            backjump(*level*, φ^p, μ^p)
        else
            $\ell \leftarrow$ *decideNextLit*(φ^p, μ^p) ▷ Next literal ℓ to split on
            level $\leftarrow$ *level* $+ 1$
    end while

CDCL($\mathcal{T}$) algorithm

- preProcess: possibly simplifies/updates φ and μ
  - Boolean pre-processing + $\mathcal{T}$-specific rewriting
    - e.g., $x < 5 \wedge x < 8 \implies x < 5, \quad x = y \wedge f(x) \neq f(y) \implies \perp$
- $\mathcal{T}2\mathcal{B}$ maps a $\mathcal{T}$-formula to its Boolean abstraction ($\mathcal{B}2\mathcal{T} = \mathcal{T}2\mathcal{B}^{-1}$)
  - e.g., $\mathcal{T}2\mathcal{B}(A \vee x + 3 < y \vee y \leq 0) = A \vee B_1 \vee B_2$
- propagate: iteratively applies unit propagation, $\mathcal{T}$-consistency checks and $\mathcal{T}$-propagation. It updates φ^p, μ^p and returns either:
  - SAT: if $\mu^p \models_p \varphi^p$ and $\mu \models_{\mathcal{T}} \varphi$
  - UNSAT: if $\mu^p \wedge \varphi^p \models_p \perp$ or $\mu \wedge \varphi \models_{\mathcal{T}} \perp$
  - UNKNOWN: if no more literals can be deduced (fixpoint)
- decideNextLit: select the next literal to split on according to given heuristics as in standard DPLL (but $\mathcal{T}$-information possibly exploited)

CDCL($\mathcal{T}$) algorithm

- analyzeConflict performs conflict analysis if UNSAT is returned
- If conflict detected by Boolean propagation ($\mu^p \wedge \varphi^p \models_p \bot$) a Boolean conflict set η^p is produced (see CDCL)
- If conflict detected by $\mathcal{T}$-propagation ($\mu \wedge \varphi \models_{\mathcal{T}} \bot$) a theory conflict set η is produced and abstracted to η^p
- Then, φ^p updated with $\neg\eta^p \wedge \varphi^p$ and a decision level is returned:
  - If level < 0, no more decisions are possible: φ unsatisfiable
  - Otherwise, backjump to specified level
    - "standard" DPLL does chronological backtracking: back to last level

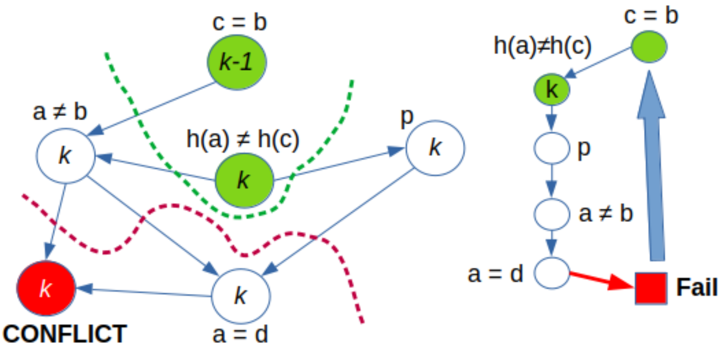
CDCL($\mathcal{T}$) conflict example

- Let $(h(a) = h(c) \vee p) \wedge (a = b \vee \neg p \vee a = d) \wedge (a \neq d \vee a = b)$ be part of a formula φ and decision $c = b \in \mu$. Consider the following:
- Decide $h(a) \neq h(c)$
- UnitPropagate p due to clause $h(a) = h(c) \vee p$
- $\mathcal{T}$-propagate $a \neq b$ because $\{c = b, h(a) \neq h(c)\} \models_{\mathcal{T}} a \neq b$
  - If $a = b$, then $c = b$ would imply $h(a) = h(c)$
- UnitPropagate $a = d$ due to clause $a = b \vee \neg p \vee a = d$
- Conflict: $a \neq d$ and $a = d$

Conflict analysis

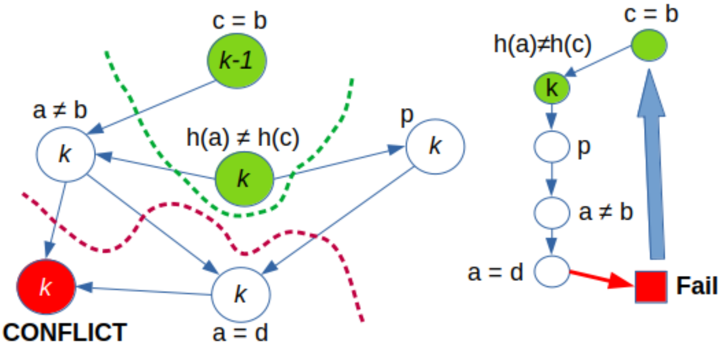
- As done in SAT, to derive an explanation from a conflict an implication graph is built
- Nodes: either decisions, derived literals or conflicts
- Edges: if $\{v_1, \ldots, v_k\} \models w$ (via unit/theory propagation) then edges $v_1 \to w, \ldots, v_k \to w$ belong to the graph
  - Note: nodes $v_1, \ldots, v_k$ might be at different decision levels
- Every cut of the graph separating sources (decisions) from the sink (the conflict) is a valid conflict clause

Implication graph



- How to cut? Typically, 1UIP clause is chosen
- UIP (Unique Implication Point) = node traversed by all paths from current decision node to conflict. 1UIP = "closest" UIP to conflict

CDCL($\mathcal{T}$) conflict example

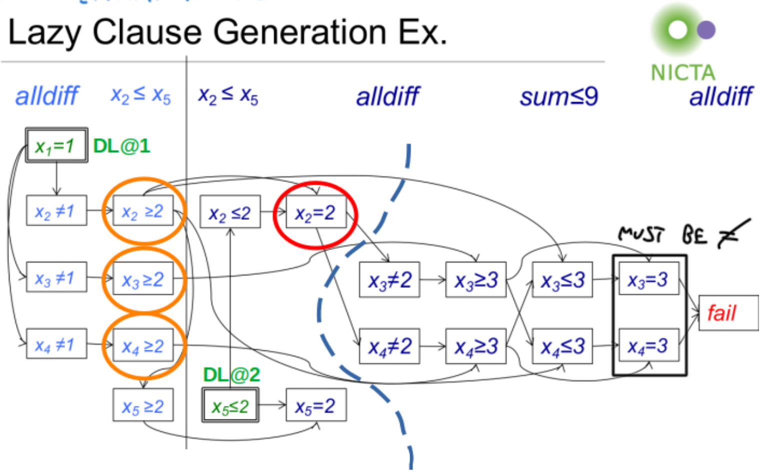


- Here, the 1UIP is $h(a) \neq h(c)$
- Conflict set is $\eta = \{h(a) \neq h(c), c = b\}$, so $h(a) = h(c) \vee c \neq b$ is added to φ and we backjump to highest point where one literal of η is not assigned

Digression: CP with LCG

DECISION EXPLANATIONS

Lazy Clause Generation Ex.



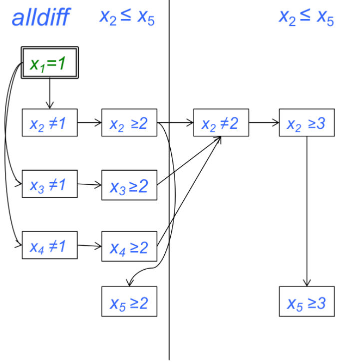
Suppose $x_1, \ldots, x_4 \in \{1..4\}$. 1UIP for level 2 is $[\![x_2 = 2]\!]$. Conflict set is $\eta = \{[\![x_2 \geq 2]\!], [\![x_3 \geq 2]\!], [\![x_4 \geq 2]\!], [\![x_2 = 2]\!]\}$. Backjump to DL@1

Example from a talk by Prof. Peter J. Stuckey.

Digression: CP with LCG

Backjumping

NICTA



- Backtrack to second last level in nogood
- Nogood will propagate
- Note stronger domain than usual backtracking
  - $D(x_2) = \{3..4\}$

$\{x_2 \geq 2, x_3 \geq 2, x_4 \geq 2, x_2 = 2\} \rightarrow$ *false*

Example from a talk by Prof. Peter J. Stuckey.

Exercise

- Exercise: simulate the execution of $\mathcal{T}$-CDCL($\varphi, \emptyset$) with $\varphi \equiv g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$
  - What is φ^p in this case?
  - How many backjumps?
  - Draw the implication graph

Exercise

- Exercise: simulate the execution of $\mathcal{T}$-CDCL$(\varphi, \emptyset)$ with $\varphi \equiv g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$
  - What is φ^p in this case? $\ell_1 \wedge (\neg \ell_2 \vee \ell_3) \wedge \neg \ell_4$
  - How many backjumps?
  - Draw the implication graph

Exercise

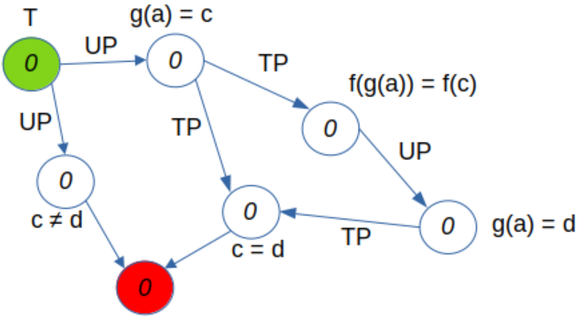
- Exercise: simulate the execution of $\mathcal{T}$-CDCL$(\varphi, \emptyset)$ with $\varphi \equiv g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$
  - What is φ^p in this case? $\ell_1 \wedge (\neg \ell_2 \vee \ell_3) \wedge \neg \ell_4$
  - How many backjumps? 0
  - Draw the implication graph

Exercise

- Exercise: simulate the execution of $\mathcal{T}$-CDCL$(\varphi, \emptyset)$ with $\varphi \equiv g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$
  - What is φ^p in this case? $\ell_1 \wedge (\neg\ell_2 \vee \ell_3) \wedge \neg\ell_4$
  - How many backjumps? 0
  - Draw the implication graph

Exercise

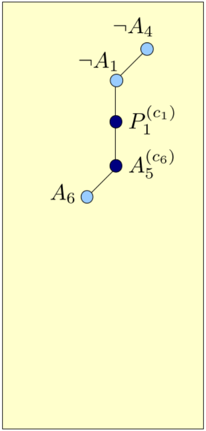
- Exercise: simulate the execution of $\mathcal{T}$-CDCL$(\varphi, \emptyset)$ with $\varphi \equiv g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$
  - What is φ^p in this case? $\ell_1 \wedge (\neg \ell_2 \vee \ell_3) \wedge \neg \ell_4$
  - How many backjumps? 0
  - Draw the implication graph



CDCL($\mathcal{T}_{\mathbb{Z}}$) example

	$\varphi \stackrel{\text{def}}{=}$	$\varphi^{\text{Bool}} \stackrel{\text{def}}{=}$
c_1 :	$(2x_2 - x_3 > 2) \vee P_1$	$A_1 \vee P_1$
c_2 :	$\neg P_2 \vee (x_1 - x_5 \leq 1)$	$\neg P_2 \vee A_2$
c_3 :	$\neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2$	$\neg A_3 \vee \neg P_2$
c_4 :	$\neg(3x_1 - x_3 \leq 6) \vee \neg P_1$	$\neg A_4 \vee \neg P_1$
c_5 :	$P_1 \vee (3x_1 - 2x_2 \leq 3)$	$P_1 \vee A_3$
c_6 :	$(x_2 - x_4 \leq 6) \vee \neg P_1$	$A_5 \vee \neg P_1$
c_7 :	$P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2$	$P_1 \vee A_6 \vee \neg P_2$
c_8 :	$P_2 \vee (2x_2 - 3x_1 \geq 5) \vee$ $(x_3 + x_5 - 4x_1 \geq 0)$	$P_2 \vee A_7 \vee A_8$

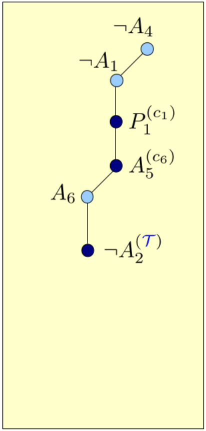
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6]$$



Example from CAV Verification Mentoring Workshop 2017 talk by Alberto Griggio (FBK, Trento). Light blue nodes = decisions, dark blue nodes = entailed literals

CDCL($\mathcal{T}_{\mathbb{Z}}$) example

$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & \\
c_2: & \neg P_2 \vee (x_1 - x_5 \le 1) & \\
c_3: & \neg(3x_1 - 2x_2 \le 3) \vee \neg P_2 & \\
c_4: & \neg(3x_1 - x_3 \le 6) \vee \neg P_1 & \\
c_5: & P_1 \vee (3x_1 - 2x_2 \le 3) & \\
c_6: & (x_2 - x_4 \le 6) \vee \neg P_1 & \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & \\
c_8: & P_2 \vee (2x_2 - 3x_1 \ge 5) \vee & \\
 & (x_3 + x_5 - 4x_1 \ge 0) &
\end{array}
\qquad
\begin{array}{ll}
\varphi^{\text{Bool}} & \stackrel{\text{def}}{=} \\
 & A_1 \vee P_1 \\
 & \neg P_2 \vee A_2 \\
 & \neg A_3 \vee \neg P_2 \\
 & \neg A_4 \vee \neg P_1 \\
 & P_1 \vee A_3 \\
 & A_5 \vee \neg P_1 \\
 & P_1 \vee A_6 \vee \neg P_2 \\
 & P_2 \vee A_7 \vee A_8
\end{array}
$$



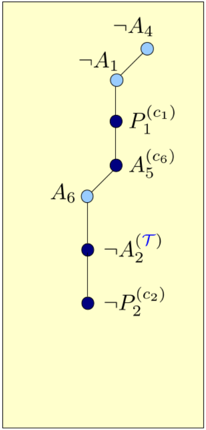
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6]$$

$$\dfrac{\dfrac{\neg(3x_1 - x_3 \le 6) \qquad (x_3 = 3x_5 + 4)}{\neg(3x_1 - 3x_5 \le 10)}}{\neg(x_1 - x_5 \le 1) \equiv \neg A_2}$$

CDCL($\mathcal{T}_\mathbb{Z}$) example

$$
\begin{array}{lll@{\qquad}ll}
\varphi & \stackrel{\text{def}}{=} & & \varphi^{\text{Bool}} & \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & & & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & & & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & & & \neg A_3 \vee \neg P_2 \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & & & \neg A_4 \vee \neg P_1 \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & & & P_1 \vee A_3 \\
c_6: & (x_2 - x_4 \leq 6) \vee \neg P_1 & & & A_5 \vee \neg P_1 \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & & & P_1 \vee A_6 \vee \neg P_2 \\
c_8: & P_2 \vee (2x_2 - 3x_1 \geq 5) \vee & & & P_2 \vee A_7 \vee A_8 \\
 & (x_3 + x_5 - 4x_1 \geq 0) & & &
\end{array}
$$

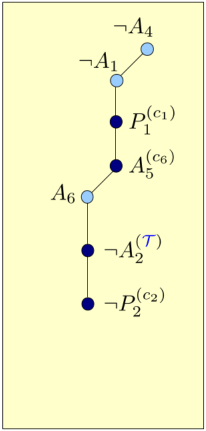
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2]$$



CDCL($\mathcal{T}_{\mathbb{Z}}$) example

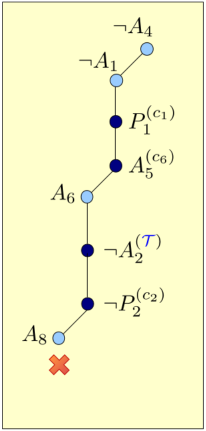
$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & \\
c_6: & (x_2 - x_4 \leq 6) \vee \neg P_1 & \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & \\
c_8: & P_2 \vee (2x_2 - 3x_1 \geq 5) \vee & \\
 & (x_3 + x_5 - 4x_1 \geq 0) &
\end{array}
\qquad
\begin{array}{ll}
\varphi^{\text{Bool}} & \stackrel{\text{def}}{=} \\
A_1 \vee P_1 & \\
\neg P_2 \vee A_2 & \\
\neg A_3 \vee \neg P_2 & \\
\neg A_4 \vee \neg P_1 & \\
P_1 \vee A_3 & \\
A_5 \vee \neg P_1 & \\
P_1 \vee A_6 \vee \neg P_2 & \\
P_2 \vee A_7 \vee A_8 &
\end{array}
$$

$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2]$$



CDCL($\mathcal{T}_{\mathbb{Z}}$) example

	$\varphi \stackrel{\text{def}}{=}$	$\varphi^{\text{Bool}} \stackrel{\text{def}}{=}$
$c_1:$	$(2x_2 - x_3 > 2) \vee P_1$	$A_1 \vee P_1$
$c_2:$	$\neg P_2 \vee (x_1 - x_5 \le 1)$	$\neg P_2 \vee A_2$
$c_3:$	$\neg(3x_1 - 2x_2 \le 3) \vee \neg P_2$	$\neg A_3 \vee \neg P_2$
$c_4:$	$\neg(3x_1 - x_3 \le 6) \vee \neg P_1$	$\neg A_4 \vee \neg P_1$
$c_5:$	$P_1 \vee (3x_1 - 2x_2 \le 3)$	$P_1 \vee A_3$
$c_6:$	$(x_2 - x_4 \le 6) \vee \neg P_1$	$A_5 \vee \neg P_1$
$c_7:$	$P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2$	$P_1 \vee A_6 \vee \neg P_2$
$c_8:$	$P_2 \vee (2x_2 - 3x_1 \ge 5) \vee (x_3 + x_5 - 4x_1 \ge 0)$	$P_2 \vee A_7 \vee A_8$



$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2, A_8]$$

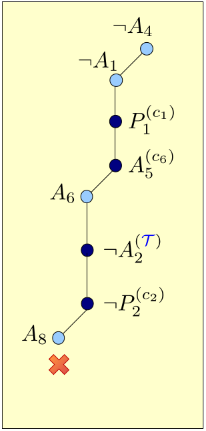
$$\frac{\dfrac{\neg(3x_1 - x_3 \le 6) \qquad \neg(x_1 - x_5 \le 1)}{\neg(-x_3 + 3x_5 \le 3)} \qquad (x_3 + x_5 - 4x_1 \ge 0)}{\bot}$$

CDCL($\mathcal{T}_{\mathcal{Z}}$) example

$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \qquad\qquad \varphi^{\text{Bool}} \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \le 1) & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \le 3) \vee \neg P_2 & \neg A_3 \vee \neg P_2 \\
c_4: & \neg(3x_1 - x_3 \le 6) \vee \neg P_1 & \neg A_4 \vee \neg P_1 \\
c_5: & P_1 \vee (3x_1 - 2x_2 \le 3) & P_1 \vee A_3 \\
c_6: & (x_2 - x_4 \le 6) \vee \neg P_1 & A_5 \vee \neg P_1 \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & P_1 \vee A_6 \vee \neg P_2 \\
c_8: & P_2 \vee (2x_2 - 3x_1 \ge 5) \vee & P_2 \vee A_7 \vee A_8 \\
 & (x_3 + x_5 - 4x_1 \ge 0) &
\end{array}
$$

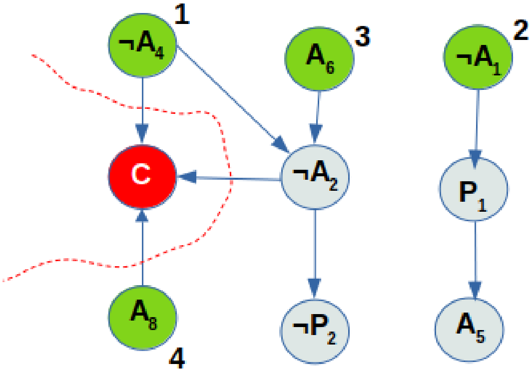
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2, A_8]$$

$$
\dfrac{\dfrac{\neg(3x_1 - x_3 \le 6) \qquad \neg(x_1 - x_5 \le 1)}{\neg(-x_3 + 3x_5 \le 3)} \qquad (x_3 + x_5 - 4x_1 \ge 0)}{\bot}
$$



Exercise: write the implication graph, the 1UIP and the conflict set

CDCL($\mathcal{T}_\mathcal{Z}$) example



Exercise: 1UIP $= A_8$, conflict set $\eta = \{A_8, A_6, \neg A_4\}$.
Backjump to DL 3 and add $\neg A_8 \vee \neg A_6 \vee A_4$. This unit propagates $\neg A_8$...

SMT-LIB Encoding

```
(declare-const x1 Int)
(declare-const x2 Int)
(declare-const x3 Int)
(declare-const x4 Int)
(declare-const x5 Int)
(declare-const P1 Bool)
(declare-const P2 Bool)
; (2x2 - x3 > 2) \/ P1
(assert (or (> (- (* 2 x2) x3) 2) P1))
; ~P2 \/ x1 - x5 <= 1
(assert (or (not P2) (<= (- x1 x5) 1)))
   ...
   ...
   ...
(check-sat)
(get-model)
```

ex_lia.smt2

SMT-LIB Encoding

```
(declare-const x1 Int)
(declare-const x2 Int)
(declare-const x3 Int)
(declare-const x4 Int)
(declare-const x5 Int)
(declare-const P1 Bool)
(declare-const P2 Bool)
; (2x2 - x3 > 2) \/ P1
(assert (or (> (- (* 2 x2) x3) 2) P1))
; ~P2 \/ x1 - x5 <= 1
(assert (or (not P2) (<= (- x1 x5) 1)))
    ...
    ...
    ...
(check-sat)
(get-model)
```

Is this satisfiable?

SMT-LIB Encoding

```
sat
(
  (define-fun x3 () Int
    (- 3))
  (define-fun P2 () Bool
    true)
  (define-fun x2 () Int
    0)
  (define-fun x1 () Int
    2)
  (define-fun x5 () Int
    1)
  (define-fun x4 () Int
    0)
  (define-fun P1 () Bool
    true)
)
```

$$
\begin{aligned}
c_1 &: \quad (2x_2 - x_3 > 2) \vee P_1 \\
c_2 &: \quad \neg P_2 \vee (x_1 - x_5 \leq 1) \\
c_3 &: \quad \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 \\
c_4 &: \quad \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 \\
c_5 &: \quad P_1 \vee (3x_1 - 2x_2 \leq 3) \\
c_6 &: \quad (x_2 - x_4 \leq 6) \vee \neg P_1 \\
c_7 &: \quad P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 \\
c_8 &: \quad P_2 \vee (2x_2 - 3x_1 \geq 5) \vee \\
&\quad\;\; (x_3 + x_5 - 4x_1 \geq 0)
\end{aligned}
$$

CDCL($\mathcal{T}$) vs CDCL

- Summarizing, the main extensions of CDCL($\mathcal{T}$) w.r.t. CDCL are:
- $\mathcal{T}$-propagation
    - in addition to Boolean propagation
- $\mathcal{T}$-conflicts or mixed $\mathcal{T}$-conflicts
    - in addition to Boolean conflicts
- Cheap operations computed first: if $\mathcal{T}$-solving is computationally expensive, calls to $\mathcal{T}$-solver can be delayed

Take-home messages

- SMT solving is strongly coupled to SAT solving
- Two orthogonal approaches: eager vs lazy encoding of SMT$\rightarrow$SAT
- Eager approach: translates upfront a SMT formula to equisatisfiable SAT formula (a.k.a. "bit-blasting")
  - No need of theory solvers, we can use best available SAT solver(s)
  - Complex, *ad hoc* encodings needed for all the theories we use
  - Examples: small-domain encoding, direct encoding

Take-home messages

- Lazy approach: combine SAT solvers + $\mathcal{T}$-solvers
  - $\mathcal{T}$-information used lazily over Boolean abstractions
  - Everyone (SAT and SMT solvers) does what she is good at
  - Modular and flexible
  - Typically, but not necessarily, more efficient than eager approach
    - Bit-blasting commonly used for bit-vectors
- CDCL($\mathcal{T}$): well-established lazy approach. Extends CDCL with:
  - theory propagation
  - theory conflicts
  - sometimes called DPLL($\mathcal{T}$)

Resources

- Handbook of Satisfiability – Chapter 12 "*Satisfiability Modulo Theories*" by C. Barrett, R. Sebastiani, S.A. Seshia, C. Tinelli
  - Search "Satisfiability Modulo Theories - EECS at UC Berkeley"
- Barrett, Clark, and Cesare Tinelli. "Satisfiability modulo theories." Handbook of model checking. Springer, Cham, 2018. 305-343.
- SAT/SMT schools
  - `https://sat-smt.in/`
- ...