3. Theory solvers, combinations and extensions

Prof. Roberto Amadini

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

Combinatorial Decision Making and Optimization

2nd cycle degree programme in Artificial Intelligence
University of Bologna, Academic Year 2022/23

Theory solvers

- So far, we introduced the basics of SMT solving without focusing much on background theories and their solvers
  - Eager vs lazy approaches
- In its simplest form, a $\mathcal{T}$-solver takes as input a conjunction of $\mathcal{T}$-literals μ and decides whether μ is $\mathcal{T}$-satisfiable
- We can see a SMT solver as a "collection" of theory solvers
- What are the crucial features for a $\mathcal{T}$solver?

Theory solvers

- Early pruning: invoke $\mathcal{T}$-solver on partial assignments during search
  - Can drastically reduce the search, possibly many $\mathcal{T}$-solver calls
- Incrementality: when a new constraint is added, no need to redo all the computation "from scratch"
- Backtrackability: support cheap (stack-based) removal of constraints without "resetting" the internal state

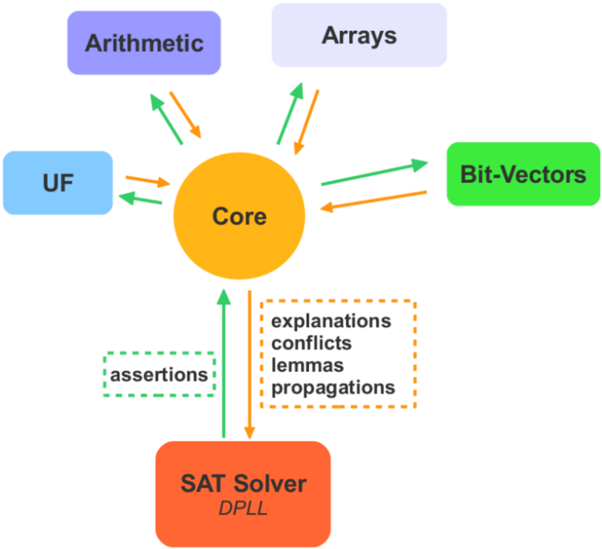
Theory solvers

- Literal deduction: $\mathcal{T}$-solver can perform deductions of literals not yet assigned in the input formula ($\mathcal{T}$-propagation)
- Explanation generation: when a conflict involving a literal ℓ is found, is necessary to have a (possibly short) explanation $\ell_1 \wedge \cdots \wedge \ell_n \rightarrow \ell$ to perform conflict analysis and backjumping

What theories?

- Uninterpreted functions (EUF)
- Arithmetic
  - LIA, LRA, LIRA, . . .
- Arrays
- Bit-vectors
- Strings
- . . .

What theories?



EUF theory

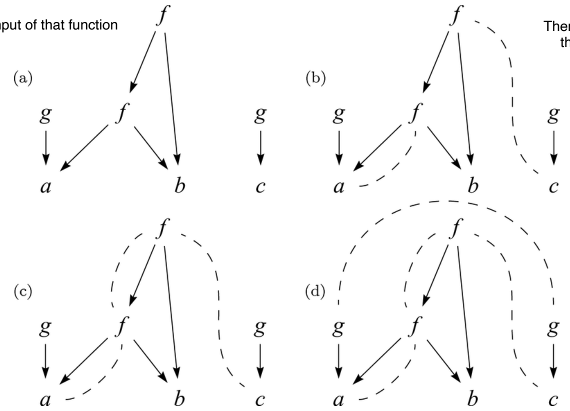
- Simplest theory: all Σ-models of a given signature Σ
- Conjuctions of literals of $\mathcal{T}_{\mathcal{E}}$ decided in polynomial time with congruence closure procedures:
  - Add fresh c and replace each $p(t_1, \ldots, t_k)$ with $f_p(t_1, \ldots, t_k) = c$
  - Partition input literals into equalities (E) and disequalities (D)
  - Let E^* be the congruence closure of E, i.e., the smallest equivalence relation $\equiv_E$ over the terms of E such that:
    - $t_1 = t_2 \in E \implies t_1 \equiv_E t_2$
    - For each $f(s_1, \ldots, s_k), f(t_1, \ldots, t_k)$ occurring in E, if $s_i \equiv_E t_i$ for each $i \in \{1, \ldots, k\}$ then $f(s_1, \ldots, s_k) \equiv_E f(t_1, \ldots, t_k)$ (congruence property)
  - Then, Φ satisfiable iff for each $t_1 \neq t_2 \in D$, $t_1 \not\equiv_E t_2$
- Standard algorithms use a DAG to represent terms, and union-find data structure (a.k.a. merge-find or disjoint-set) for the classes of $\equiv_E$

EUF theory

Example*: $\phi \;\equiv\; f(a,b) = a \;\wedge\; f(f(a,b),b) = c \;\wedge g(a) \neq g(c)$

put an arrow for each input of that function

Then put a dashed line between the function and the output



Then put a dashed line between equal functions: f(f(a,b),b) == f(a,b) and g(a)==g(c) —> contraddiction for the last one

(a) DAG for ϕ-terms. (b) E-graph: equivalences are the equalities in ϕ. (c) $f(f(a,b),b)) \equiv_E f(a,b)$ because $f(a,b) = a$. (d) $g(a) \equiv_E g(c)$ because $a \equiv_E c$. Since $g(a) \neq g(c)$ and $g(a) \equiv_E g(c)$, ϕ is unsatisfiable

*From: D. Detlefs, G. Nelson, J.B. Saxe. "Simplify: a theorem prover for program checking." JACM, 52.3 (2005): 365-473

LRA theory

- Consider LRA = Linear Real Arithmetic theory, having signature $\Sigma_{LRA} = (\mathbb{Q}, +, -, *, \leq)$ and linear multiplications only

- We could decide LRA-literals with Fourier-Motzkin elimination
  - Replace $t_1 \neq t_2$ with $t_1 < t_2 \vee t_2 < t_1$, and $t_1 \leq t_2$ with $t_1 < t_2 \vee t_1 = t_2$ (case splitting)
  - Eliminate equalities and apply Fourier-Motzkin elimination to all variables to determine its satisfiability
    - `https://en.wikipedia.org/wiki/Fourie-Motzkin_elimination`

- Not practical for large set of constraints, simplex method preferable

LIA theory

- Consider LIA = Linear Integer Arithmetic theory, having signature $\Sigma_{LIA} = (\mathbb{Z}, +, -, *, \leq)$ and linear multiplications only
  - if not linear, undecidable (*Peano arithmetic*)
  - if fully quantified, *Presburger arithmetic*
  - if quantifier-free, different decision procedures exist
- As for LRA, we can apply methods like Fourier-Motzkin, but Simplex + branch & bound/cut generally better
- Methods exist also for LIRA = integer + real arithmetic and NLA = nonlinear arithmetic
  - E.g., `https://microsoft.github.io/z3guide/docs/theories/Arithmetic/`

Difference logic

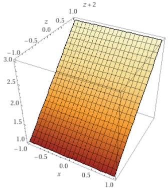
- Consider now DL = Difference Logic theory, having atomic formulas of the form $x - y \leq k$ with x, y variables and k constant
    - Constraints $x - y \bowtie k$ with $\bowtie \in \{=, \neq, <, \geq, >\}$ can be rewritten

- E.g. if $x, y \in \mathbb{Z}$:

  $x - y > 5 \;\wedge\; x = z + 2 \implies$

  $x - y \geq 6 \;\wedge\; x - z \leq 2 \;\wedge\; x - z \geq 2 \implies$

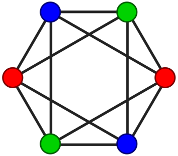
  $y - x \leq -6 \;\wedge\; x - z \leq 2 \;\wedge\; z - x \leq -2$



- Unary constraints $x \leq k$ can be rewritten into $x + z_0 \leq k$ by enforcing $z_0 = 0$ in any satisfying assignment

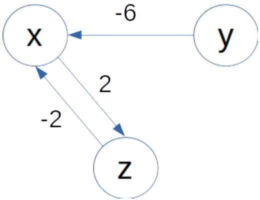
Difference logic and k-coloring

- If we allow $\neq$ and the domain is $\mathbb{Z}$, deciding satisfiability of DL formulas is NP-hard, e.g., it is "as hard as" k-coloring problem
  - If we have k colors available, can we color a graph s.t. adjacent nodes have different colors? If $k \geq 3$ the problem is NP-hard
- Formally, given graph (V, E) and $k \in \mathbb{N}$, does it exist a function $c : V \to \{1, \ldots, k\}$ s.t. for each $(i, j) \in E$ we have $c(i) \neq c(j)$?
- Any k-coloring instance can be mapped to a DL formula with $|V|$ variables, $|E|$ disequalities $x_i \neq x_j$ for each $(i, j) \in E$ and $2|V|$ disequalities $1 \leq x_i \leq k$
  - If we can decide the DL formula in polynomial time, we can solve any problem of NP in polynomial time



Difference logic as graph problem

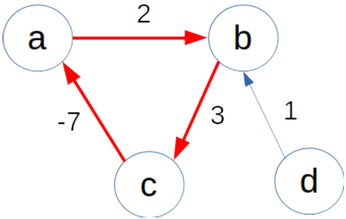
- From DL literals in Φ we can get a directed weighted graph $\mathcal{G}_\Phi$ with:
  - a node for each variable occuring in Φ
  - a weighted edge $x \xrightarrow{k} y$ for each $x - y \leq k \in \Phi$
- E.g., if $\Phi = \{y - x \leq -6,\ x - z \leq 2,\ z - x \leq -2\}$ then $\mathcal{G}_\Phi$ is:



- Theorem: Φ is inconsistent $\iff$ $\mathcal{G}_\Phi$ has a negative cycle

Difference logic as graph problem

- *Example*: let $\Phi = \{a - b \leq 2, b - c \leq 3, c - a \leq -7, d - b \leq 1\}$. The $\mathcal{G}_\Phi$ graph is:



- Negative loop $a \xrightarrow{2} b \xrightarrow{3} c \xrightarrow{-7} a$ (total weight -2): Φ inconsistent
  - $a \geq c + 7$ conflicts with $a \leq b + 2 \leq (c + 3) + 2 = c + 5$

Equality (==) is <= and >=

Difference logic as graph problem

- Negative loops can be detected with Bellman-Ford in $O(|V||E|)$ by adding to V a source vertex x_0 and an edge $x_0 \xrightarrow{0} x$ for each $x \in E$
  - `https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm`
  - Other more efficient variant exists

- Negative loops denotes inconsistency explanations
  - Not minimal in general

- Theory propagations computed from consistent graphs: if there is a path between x and y with total weight k, we can deduce $x - y \leq k$
  - If $x \xrightarrow{k_1} x_1 \xrightarrow{k_2} x_2 \xrightarrow{k_3} \dots \xrightarrow{k_n} y$ the total weight is $k = \sum_{i=1}^{n} k_i$ and $x - x_1 \leq k_1, x_1 - x_2 \leq k_2, \dots, x_n - y \leq k_n$ hence $(x - x_1) + (x_1 - x_2) + \dots + (x_n - y) \leq \sum_{i=1}^{n} k_i = k$ thus $x + (-x_1 + x_1) + \dots + (-x_n + x_n) + y \leq k$, i.e., $x - y \leq k$

Other theories

- Bit-vectors: typically BV formulas are fist simplified, and then encoded into SAT formulas (bit-blasting)
- Arrays: typically theory axioms instantiation + congruence closure and optimizations
- (Multi)-sets
- Strings
- Floating points
- ...

Combining Theories

Need for combination

- So far we considered theories individually. But often SMT formulas contain atoms from disparate theories
- In particular software verification applications can generate constraints over several data types
  - integers, floating points, bit-vectors, arrays, strings, ...
- E.g., formula $a = b + 2 \wedge A = write(B, a + 1, 4) \wedge (f(a) \vee g(b + 1))$ involves theory of linear arithmetic, arrays, and EUF
- Given $\mathcal{T}_i$-solvers for theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, can we combine them to get a solver for $\bigcup_i \mathcal{T}_i$?

Example

- Consider formula $f(f(x) - f(y)) = a \wedge f(0) = a + 2 \wedge x = y$
  - Two theories involved: EUF and linear arithmetic (LA)
- 1st step: purification. Each literal must belong to only one theory
  - Fresh constants needed: $e_1, \ldots, e_5$
- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA}, \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
  $\underbrace{e_4 = 0, e_5 = a + 2}_{LA}, \underbrace{f(e_4) = e_5, x = y}_{EUF}$
- In this way EUF and LA solvers only share $a, e_1, \ldots, e_5$
- To merge the corresponding models, solvers must agree on equalities between shared constants, a.k.a. interface equalities

Example

- 2^{nd} step: satisfiability check and equalities exchange
- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA},\ \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
  $\underbrace{e_4 = 0, e_5 = a + 2}_{LA},\ \underbrace{f(e_4) = e_5, x = y}_{EUF}$
- Both EUF-solver and LA-solver say SAT
- EUF solver deduces that $\{x = y, f(x) = e_2, f(y) = e_3\} \models e_2 = e_3$ and sends the literal to the LA solver

Example

- 2nd step: satisfiability check and equalities exchange

- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA}$, $\underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF}$,
$\underbrace{e_4 = 0, e_5 = a + 2, e_2 = e_3}_{LA}$, $\underbrace{f(e_4) = e_5, x = y}_{EUF}$

- Both EUF-solver and LA-solver say SAT

- EUF solver deduces that $\{x = y, f(x) = e_2, f(y) = e_3\} \models e_2 = e_3$ and sends the literal to the LA solver

Example

- 2nd step: satisfiability check and equalities exchange

- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA}, \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
$\underbrace{e_4 = 0, e_5 = a + 2, e_2 = e_3}_{LA}, \underbrace{f(e_4) = e_5, x = y}_{EUF}$

- Both EUF-solver and LA-solver say SAT

- LA solver deduces that $\{e_2 - e_3 = e_1, e_4 = 0, e_2 = e_3\} \models e_1 = e_4$ and sends the literal to the EUF solver

Example

- 2nd step: satisfiability check and equalities exchange
- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA},\ \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
$\underbrace{e_4 = 0, e_5 = a + 2, e_2 = e_3}_{LA},\ \underbrace{f(e_4) = e_5, x = y, e_1 = e_4}_{EUF}$
- Both EUF-solver and LA-solver say SAT
- LA solver deduces that $\{e_2 - e_3 = e_1, e_4 = 0, e_2 = e_3\} \models e_1 = e_4$ and sends the literal to the EUF solver

Example

- 2^{nd} step: satisfiability check and equalities exchange

- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA}, \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
$\underbrace{e_4 = 0, e_5 = a + 2, e_2 = e_3}_{LA}, \underbrace{f(e_4) = e_5, x = y, e_1 = e_4}_{EUF}$

- Both EUF-solver and LA-solver say SAT

- EUF solver deduces that $\{f(e_1) = a, f(e_4) = e_5, e_1 = e_4\} \models a = e_5$ and sends the literal to the LA solver

Example

- 2nd step: satisfiability check and equalities exchange
- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA}, \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
  $\underbrace{e_4 = 0, e_5 = a + 2, e_2 = e_3, a = e_5}_{LA}, \underbrace{f(e_4) = e_5, x = y, e_1 = e_4}_{EUF}$
- Both EUF-solver and LA-solver say SAT
- EUF solver deduces that $\{f(e_1) = a, f(e_4) = e_5, e_1 = e_4\} \models a = e_5$ and sends the literal to the LA solver

Example

- 2^{nd} step: satisfiability check and equalities exchange

- Purified formula: $\underbrace{e_1 = e_2 - e_3}_{LA}, \underbrace{f(e_1) = a, e_2 = f(x), e_3 = f(y)}_{EUF},$
$\underbrace{e_4 = 0, e_5 = a + 2, e_2 = e_3, a = e_5}_{LA}, \underbrace{f(e_4) = e_5, x = y, e_1 = e_4}_{EUF}$

- EUF-solver say SAT...

- ...but LA-solver say UNSAT: $\{e_5 = a + 2, a = e_5\} \models \perp$

- Hence the original formula is UNSAT

Nelson-Oppen procedure (convex case)

- Let Σ_1, Σ_2 be signatures and $\mathcal{T}_1, \mathcal{T}_2$ their theories. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are:
  - signature-disjoint
    - $\Sigma_1 \cap \Sigma_2 = \emptyset$
  - stably-infinite
    - Σ-theory $\mathcal{T}$ of sort σ is stably infinite if every T-satisfiable Σ-formula has a model interpreting σ as an infinite set
  - convex
    - For each set of $\mathcal{T}_i$-literals S we have that $\mathcal{S} \models_{\mathcal{T}_i} a_1 = b_1 \vee \cdots \vee a_n = b_n$ implies that $S \models a_k = b_k$ for some $k \in \{1, \ldots, n\}$
- then we can check the $(T_1 \cup T_2)$-satisfiability with the deterministic Nelson-Oppen algorithm

Nelson-Oppen procedure (convex case)

Let S be a $(T_1 \cup T_2)$-formula and E the set of interface equalities between S_1 and S_2. Deterministic Nelson-Oppen steps:

1. Purify S and split it into $\mathcal{S}_1$ and $\mathcal{S}_2$
   - $\mathcal{S}_i$ contains $\mathcal{T}_i$-literals only
2. If $\mathcal{S}_1 \models_{\mathcal{T}_1} \bot$, then return UNSAT
3. If $\mathcal{S}_2 \models_{\mathcal{T}_2} \bot$, then return UNSAT
4. If $\mathcal{S}_1 \models_{\mathcal{T}_1} x = y$ with $x = y \in E - S_2$, then $S_2 \leftarrow S_2 \cup \{x = y\}$ and go to 3
5. If $\mathcal{S}_2 \models_{\mathcal{T}_2} x = y$ with $x = y \in E - S_1$, then $S_1 \leftarrow S_1 \cup \{x = y\}$ and go to 2
6. return SAT

Nelson-Oppen procedure (non-convex case)

- Why we needed convex theories? Consider the following formula:

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

involving linear integer arithmetic (LIA) and EUF theories

- Let's purify the formula by introducing the interface equalities:

$$e_1 = 1,\ e_2 = 2,\ e_3 = f(e_2),\ e_4 = f(e_1),\ e_3 = e_4 + 3$$

Nelson-Oppen procedure (non-convex case)

- Now let's check satisfiability and exchange entailed equalities:

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

- Both EUF-solver and LIA-solver say SAT
- EUF solver deduces that $\{f(e_1) = a, f(e_1) = e_4\} \models a = e_4$ and sends the literal to the LA solver

Nelson-Oppen procedure (non-convex case)

- Now let's check satisfiability and exchange entailed equalities:

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

- Both EUF-solver and LIA-solver say SAT
- EUF solver deduces that $\{f(e_1) = a, f(e_1) = e_4\} \models a = e_4$ and sends the literal to the LA solver

Nelson-Oppen procedure (non-convex case)

- Now let's check satisfiability and exchange entailed equalities:

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

- Both EUF-solver and LIA-solver say SAT
- EUF and LIA theories cannot deduce any other interface equality
  - ...but LIA solver can deduce $x = e_1 \vee x = e_2$

Nelson-Oppen procedure (non-convex case)

- Now let's check satisfiability and exchange entailed equalities:

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

- If $x = e_1$, EUF would deduce $a = b$: UNSAT
- If $x = e_2$, EUF would deduce $b = e_3$: UNSAT

Nelson-Oppen procedure (non-convex case)

- Now let's check satisfiability and exchange entailed equalities:

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

- Hence, $x = e_1 \vee x = e_2$ is false and the original formula UNSAT
- ...But we can't infer this with deterministic Nelson-Oppen procedure!

Non-deterministic Nelson-Oppen

- Deterministic Nelson-Oppen procedure doesn't work in the example above because $\mathcal{T}_{LIA}$ is not convex: $1 \leq x \leq 2 \models x = 1 \lor x = 2$ but: $1 \leq x \leq 2 \not\models x = 1$ and $1 \leq x \leq 2 \not\models x = 1$

- However, there is a non-deterministic Nelson-Oppen procedure that also works on non-convex theories
  - We still need disjoint and stably-infinite theories

- It works through arrangements of shared constants, basically doing case splitting $x = y \lor x \neq y$ between pair of shared constants x, y
  - Unsurprisingly, exponential worst-case time complexity

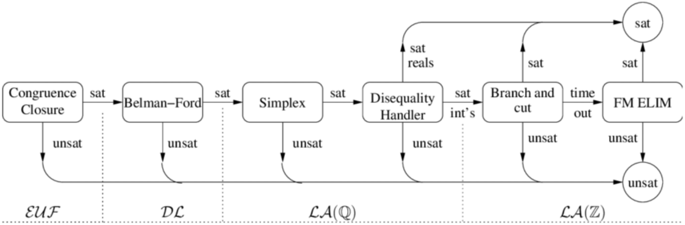
SMT Extensions

Extensions

- There are several extensions and enhancements to the SMT framework seen so far, e.g.
- Quantified formulas
- Layered solvers
- On-demand solvers
- Optimization Modulo Theory

Layered solvers

- Hierarchical approach: the problem is stratified in layers $L_1, L_2, \ldots$ of increasing complexity, solved by solvers of increasing expressiveness (but decreasing performance)
  - if solver finds conflict at L_i $\rightarrow$ use conflict to prune search space
  - otherwise activate next, more expensive solver
  - if a (partial) assignment is found, it is passed to L_{i+1}



Picture from "Lazy Satisfiability Modulo Theories." R. Sebastiani (2007)

Case splitting

- Sometimes, consistency checking requires case reasoning
- E.g., $\underbrace{read(write(A, i, x), j) \neq x}_{\ell_1} \;\wedge\; \underbrace{read(write(A, i, x), j) \neq read(A, j)}_{\ell_2}$
- This formula is unsatisfiable:
  - case $i = j$: ℓ_1 rewrites into $x \neq x$
  - case $i \neq j$: ℓ_2 rewrites into $read(A, j) \neq read(A, j)$
- A complete $\mathcal{T}_{\mathcal{A}}$-solver can "internally" detect inconsistency via case splitting and backtracking
- Alternative: lift case reasoning from $\mathcal{T}$-solver to SAT solver

Case splitting

- "On-demand" approach: $\mathcal{T}$-solver encodes the splits as clauses and sends them to SAT engine
- E.g., DPLL($\mathcal{T}_{\mathcal{A}}$) asks if $y = read(write(A, i, x), j)$ is satisfiable
- $\mathcal{T}_{\mathcal{A}}$-solver does not know yet, so it can "suggest" the following lemmas to SAT engine:
  - $y = read(write(A, i, x), j) \wedge i = j \rightarrow y = x$
  - $y = read(write(A, i, x), j) \wedge i \neq j \rightarrow y = read(A, j)$
- Pros: $\mathcal{T}$-solvers not necessarily complete, case reasoning coordinated by SAT solver, can be generalized to combinations of theories
- Cons: potential termination issues, specific criteria must be met to ensure soundness/completeness, performance issues

Optimization Modulo Theory

- OMT is an extension of SMT where we need to find a model for an input formula φ that is optimal w.r.t. an objective function f_{obj}
- φ refers to a theory $\mathcal{T} = \mathcal{T}_{\preceq} \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$ where
  - $\mathcal{T}_{\preceq}$ contains a predicate $\preceq$ representing a total order
  - $\bigcup_{i=1}^{n} \mathcal{T}_i$ might be empty
- The goal is finding a model $\mathcal{M}$ s.t. $\varphi^{\mathcal{M}} = \mathit{true}$ and $f_{obj}^{\mathcal{M}}$ is minimal according to $\preceq$
  - Maximizing $f_{obj} \equiv$ minimizing $-f_{obj}$
- Typically, $\preceq$ is the $\leq$ predicate over integers or reals
  - E.g. $\mathcal{T}_{\mathcal{LIRA}}$ + Nelson-Oppen $\mathcal{T}_i$

Optimization Modulo Theory

- OMT is "much younger" than SMT: first proposal in 2006
    - R. Nieuwenhuis and A. Oliveras. *On SAT Modulo Theories and Optimization Problems*. In SAT, volume 4121 of LNCS. Springer, 2006

- Nowadays different OMT proposals (see Sebastiani et al. works)
    - Max-SMT
    - Bit-vectors
    - Floating points
    - ...

- Some state-of-the-art SMT solvers natively provide OMT capabilities (Z3, OptiMathSAT) but others still don't (e.g. CVC5)

- Let's see an example by R. Sebastiani of OMT($\mathcal{LRA}$) with linear search

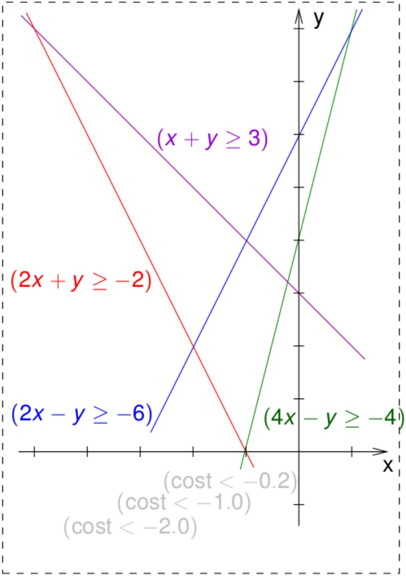
OMT($\mathcal{LRA}$) example

[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$
\begin{array}{rl}
\varphi \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\
\wedge & (\ \ A_1 \vee (x + y \geq 3)) \\
\wedge & (\neg A_2 \vee (4x - y \geq -4)) \\
\wedge & (\ \ A_2 \vee (2x - y \geq -6)) \\
\wedge & (\text{cost} < -0.2) \\
\wedge & (\text{cost} < -1.0) \\
\wedge & (\text{cost} < -2.0) \\
\text{cost} \stackrel{\text{def}}{=} & x
\end{array}
$$

- $$\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (\text{cost} < -0.2) \\ (\text{cost} < -1.0) \\ (\text{cost} < -2.0) \end{array} \right\}$$



OMT($\mathcal{LRA}$) example

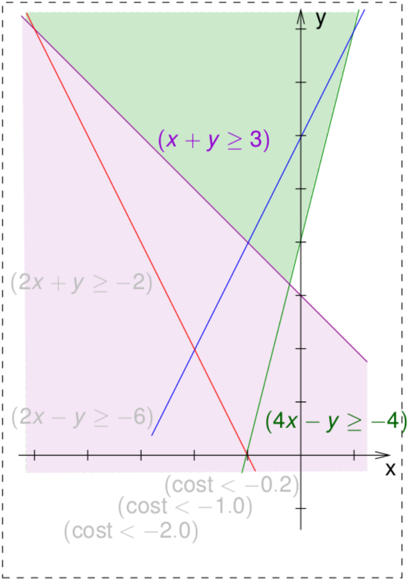
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$
\begin{array}{rll}
\varphi & \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\
& \wedge & (\ \ A_1 \vee (x + y \geq 3)) \\
& \wedge & (\neg A_2 \vee (4x - y \geq -4)) \\
& \wedge & (\ \ A_2 \vee (2x - y \geq -6)) \\
& \wedge & (\text{cost} < -0.2) \\
& \wedge & (\text{cost} < -1.0) \\
& \wedge & (\text{cost} < -2.0) \\
\text{cost} & \stackrel{\text{def}}{=} & x
\end{array}
$$

- $$\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (\text{cost} < -0.2) \\ (\text{cost} < -1.0) \\ (\text{cost} < -2.0) \end{array} \right\}$$

$\Longrightarrow$ SAT, $min = -0.2$



OMT($\mathcal{LRA}$) example

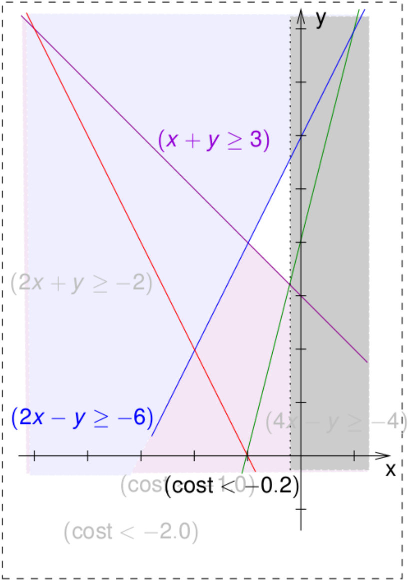
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$
\begin{array}{rcl}
\varphi & \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\
& \wedge & (\ \ A_1 \vee (x + y \geq 3)) \\
& \wedge & (\neg A_2 \vee (4x - y \geq -4)) \\
& \wedge & (\ \ A_2 \vee (2x - y \geq -6)) \\
& \wedge & (cost < -0.2) \\
& \wedge & (cost < -1.0) \\
& \wedge & (cost < -2.0) \\
cost & \stackrel{\text{def}}{=} & x
\end{array}
$$

- $\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (cost < -0.2) \\ (cost < -1.0) \\ (cost < -2.0) \end{array} \right\}$

$\Longrightarrow$ SAT, $min = -1.0$



OMT($\mathcal{LRA}$) example

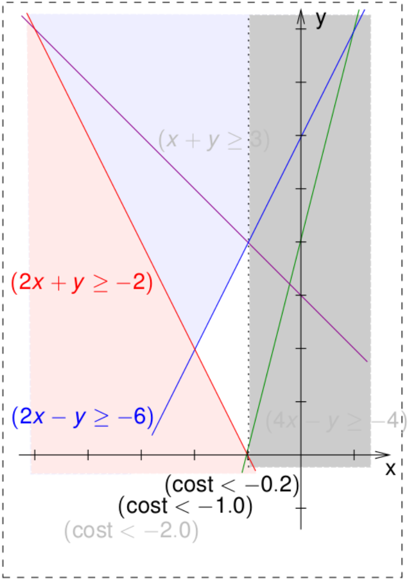
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$\begin{array}{rcl} \varphi & \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\ & \wedge & (\ A_1 \vee (x + y \geq 3)) \\ & \wedge & (\neg A_2 \vee (4x - y \geq -4)) \\ & \wedge & (\ A_2 \vee (2x - y \geq -6)) \\ & \wedge & (\text{cost} < -0.2) \\ & \wedge & (\text{cost} < -1.0) \\ & \wedge & (\text{cost} < -2.0) \\ \text{cost} & \stackrel{\text{def}}{=} & x \end{array}$$

- $\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \ A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (\text{cost} < -0.2) \\ (\text{cost} < -1.0) \\ (\text{cost} < -2.0) \end{array} \right\}$

$\Longrightarrow$ SAT, $min = -2.0$



OMT($\mathcal{LRA}$) example

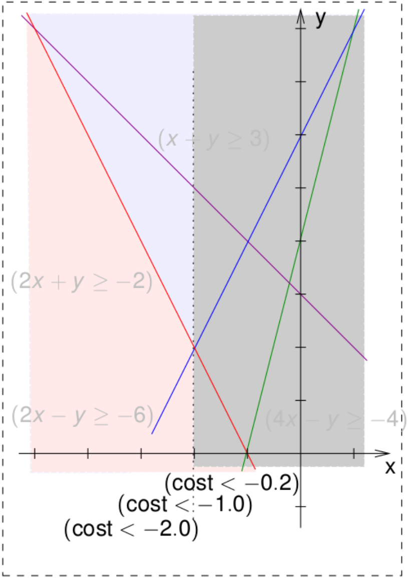
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$\begin{array}{rcl} \varphi & \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\ & \wedge & (\ \ A_1 \vee (x + y \geq 3)) \\ & \wedge & (\neg A_2 \vee (4x - y \geq -4)) \\ & \wedge & (\ \ A_2 \vee (2x - y \geq -6)) \\ & \wedge & (\text{cost} < -0.2) \\ & \wedge & (\text{cost} < -1.0) \\ & \wedge & (\text{cost} < -2.0) \\ \text{cost} & \stackrel{\text{def}}{=} & x \end{array}$$

- $\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (\text{cost} < -0.2) \\ (\text{cost} < -1.0) \\ (\text{cost} < -2.0) \end{array} \right\}$

$\Longrightarrow$ UNSAT, $min = -2.0$



Offline OMT($\mathcal{LRA}$)

- Linear search repeatedly narrows the cost domain $[l_i, u_i)$ by adding $cost < c_i$ if a model with cost c_i is found at the i-th iteration
    - If no model is found, c_i is the minimum cost
- Binary search picks a pivot $p_i \in [l_i, u_i)$ and adds $cost < p_i$
    - $p_i \simeq (l_i + u_i)/2$
    - If no model is found, look into $[p_i, u_i)$
    - On average more efficient, but we must know the cost bounds
- This approach is called offline because the SMT solvers used to find the models are black-boxes
    - No need to change their internals

Offline OMT($\mathcal{LRA}$)

Algorithm 1 Offline OMT($\mathcal{LA}(\mathbb{Q})$) Procedure based on Mixed Linear/Binary Search.

Require: $\langle \varphi, \mathsf{cost}, \mathsf{lb}, \mathsf{ub} \rangle$ {ub can be $+\infty$, lb can be $-\infty$}

$\mathsf{l} \leftarrow \mathsf{lb}$; $\mathsf{u} \leftarrow \mathsf{ub}$; $\mathsf{PIV} \leftarrow \top$; $\mathcal{M} \leftarrow \emptyset$
$\varphi \leftarrow \varphi \cup \{\neg(\mathsf{cost} < \mathsf{l}), (\mathsf{cost} < \mathsf{u})\}$
while ($\mathsf{l} < \mathsf{u}$) **do**
    if (BinSearchMode()) **then** {Binary-search Mode}
        $\mathsf{pivot} \leftarrow \mathsf{ComputePivot}(\mathsf{l}, \mathsf{u})$
        $\mathsf{PIV} \leftarrow (\mathsf{cost} < \mathsf{pivot})$
        $\varphi \leftarrow \varphi \cup \{\mathsf{PIV}\}$
        $\langle \mathsf{res}, \mu \rangle \leftarrow \mathsf{SMT.IncrementalSolve}(\varphi)$
        $\eta \leftarrow \mathsf{SMT.ExtractUnsatCore}(\varphi)$
    else {Linear-search Mode}
        $\langle \mathsf{res}, \mu \rangle \leftarrow \mathsf{SMT.IncrementalSolve}(\varphi)$
        $\eta \leftarrow \emptyset$
    end if
    if ($\mathsf{res} = \mathrm{SAT}$) **then**
        $\langle \mathcal{M}, \mathsf{u} \rangle \leftarrow \mathsf{Minimize}(\mathsf{cost}, \mu)$ *u* = current best bound
        $\varphi \leftarrow \varphi \cup \{(\mathsf{cost} < \mathsf{u})\}$
    else {$\mathsf{res} = \mathrm{UNSAT}$ }
        if ($\mathsf{PIV} \not\in \eta$) **then**   Linear search completed
            $\mathsf{l} \leftarrow \mathsf{u}$
        else
            $\mathsf{l} \leftarrow \mathsf{pivot}$
            $\varphi \leftarrow \varphi \setminus \{\mathsf{PIV}\}$ Updating binary search pivot
            $\varphi \leftarrow \varphi \cup \{\neg\mathsf{PIV}\}$
        end if
    end if
end while
return $\langle \mathcal{M}, \mathsf{u} \rangle$

From R. Sebastiani, S. Tomasi: *Optimization Modulo Theories with Linear Rational Costs*. ACM Trans. Comput. Log. 16(2): 12:1-12:43 (2015)

Inline OMT($\mathcal{LRA}$)

- The minimal cost is computed by a minimizer over linear rational inequalities
  - E.g., standard simplex techniques
- The offline approach can be improved by an inline schema
  - More efficient, but it requires modifying the internals of SMT solver
- In a nutshell, the inline approach integrates the optimization procedure into the SMT solver

Take-home messages

- Different theory solvers have been developed for different theories
  - E.g. EUF, DL, LRA, LIA, ...
- We often need to combine theories
  - Under certain conditions, Nelson-Oppen procedure can be used
- SMT solving can be optimized and extended
  - Optimization modulo theory

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/`
- ...