Satisfiability Modulo Theories

Hsin-Hung Lin

August 26, 2019

FLOLAC'19

Based on slides from SAT/SMT/AR 2019
Credits: Albert Oliveras and Bruno Dutertre

Introduction

Need of SMT

- Some problems are more naturally expressed in other logics than propositional logic
  - Software verification needs reasoning about equality, arithmetic, data structures, ...
  - First-Order Logic

- Example
  - Equality with Uninterpreted Functions (EUF)
    $g(a) = c \;\land\; \big(f\big(g(a)\big) \neq f(c) \lor g(a) = d \;\big) \;\land\; c \neq d$
  - EUF + Linear arithmetic
    $x \le y \land 2y \le x \land f(h(x) - h(y)) > f(0)$

From SAT to SMT

- SAT
  - Use propositional logic as the formalization language
  - Pros: high degree of efficiency
  - Cons: expressive but involved encodings

- SMT
  - Propositional logic + domain-specific reasoning
  - Pros: improves the expressivity
  - Cons: certain (but acceptable) loss of efficiency

SMT Problem

- Basic SMT Problem
  - Given a formula F in some logical theory T, determine whether F is satisfiable or not.
  - In addition, if F is satisfiable, provide a model of F
- DPLL(T)/CDCL(T) Approach
  - Combine a CDCL-based SAT Solver with a theory solver for T
  - The theory solver works on conjunctions of literals of T
- Combining Decision Procedures for Modularity
  - We don't want to write a global decision procedure
  - We have decision procedures for basic theories
  - We want to combine them to get a decision procedure for the combined theory.

Recall: SAT Decision procedure

- DPLL Algorithm, also called
- CDCL: Conflict-Driven-Clause-Learning

- Rules
  - Unit propagate
  - Decide
  - Fail
  - Backtrack / Backjump
  - Learning
  - Restart

DPLL – Example(1)

- $Model\ (M) \parallel Formulae(F)$
- $\emptyset \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (Decide)
- $1^{d} \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (UnitPropagate)
- $1^{d}\bar{2} \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (UnitPropagate)
- $1^{d}\bar{2}\ 3 \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (UnitPropagate)
- $1^{d}\bar{2}\ 3\ 4 \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (Backtrack)
- $\bar{1} \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (UnitPropagate)
- $\bar{1}\ 4 \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (Decide)
- $\bar{1}\ 4\ 3^{d} \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ (UnitPropagate)
- $\bar{1}\ 4\ 3^{d}\ 2 \parallel \bar{1} \lor \bar{2},\ 2 \lor 3,\ \bar{1} \lor \bar{3} \lor 4,\ 2 \lor \bar{3} \lor \bar{4},\ 1 \lor 4$ SAT

DPLL – Example(1)

- $Model\ (M)\ \|\ Formulae(F)$
- $\emptyset \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (Decide)
- $1^d \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (UnitPropagate)
- $1^d \bar{2} \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (UnitPropagate)
- $1^d \bar{2}\ 3 \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (UnitPropagate)
- $1^d \bar{2}\ 3\ 4 \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ \underline{2 \vee \bar{3} \vee \bar{4}},\ 1 \vee 4$ (Backtrack)
- $\bar{1} \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (UnitPropagate)
- $\bar{1}\ 4 \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (Decide)
- $\bar{1}\ 4\ 3^d \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ (UnitPropagate)
- $\bar{1}\ 4\ 3^d\ 2 \ \|\ \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4$ SAT

DPLL – Example(2)

- $Model\ (M) \parallel Formulae(F)$
- $\emptyset \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Decide)
- $1^d \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (UnitPropagate)
- $1^d\ 2 \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Decide)
- $1^d\ 2\ 3^d \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (UnitPropagate)
- $1^d\ 2\ 3^d\ 4 \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Decide)
- $1^d\ 2\ 3^d\ 4\ 5^d \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (UnitPropagate)
- $1^d\ 2\ 3^d\ 4\ 5^d\ \bar{6} \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Backjump)
- $1^d\ 2\ \bar{5} \parallel \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$

DPLL – Example(2)

- $Model\ (M)\ \|\ Formulae(F)$
- $\emptyset\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Decide)
- $1^d\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (UnitPropagate)
- $1^d\ 2\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Decide)
- $1^d\ 2\ 3^d\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (UnitPropagate)
- $1^d\ 2\ 3^d\ 4\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (Decide)
- $1^d\ 2\ 3^d\ 4\ 5^d\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ (UnitPropagate)
- $1^d\ 2\ 3^d\ 4\ 5^d\ \bar{6}\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ \underline{6 \vee \bar{5} \vee \bar{2}}$ (Backjump)
- $1^d\ 2\ \bar{5}\ \|\ \bar{1} \vee 2,\ \bar{3} \vee 4,\ \bar{5} \vee \bar{6},\ 6 \vee \bar{5} \vee \bar{2}$ Learned Clause $\overline{5 \wedge 2} = \bar{5} \vee \bar{2}$

Theories of Interest - EUF

- Equality with Uninterpreted Functions, i.e. "=" is equality
- Consider formula
  $a * \big(f(b) + f(c)\big) = d \ \wedge \ b * (\ f(a) + f(c))) \ \neq d \wedge a = b$
- Formula is UNSAT, but no arithmetic reasoning is needed
- If we abstract the formula into
  $h(a, g\big(f(b), f(c)\big)) = d \ \wedge h(b, g(\ f(a), f(c))) \ \neq d \wedge a = b$
- it is still UNSAT
- EUF is used to abstract non-supported constructions, e.g: Non-linear multiplication, ALUs in circuits

Theories of Interest - Arithmetic

- Bounds
  - $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$
- Difference logic
  - $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$
- UTVPI (Unit Two Variable Per Inequality)
  - $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$
- Linear arithmetic
  - e.g: $2x - 3y + 4z \leq 5$
- Non-linear arithmetic
  - e.g: $2xy + 4xz^2 - 5y \leq 10$
- Variables are either reals or integers
- Machine-inspired arithmetic
  - floating-point arithmetic

Theories of Interest - Arrays

- 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))) => a = b
- Is the following set of literals satisfiable?

  $write(a, i, x) \neq b\ \wedge read(b, i) = y\ \wedge$
  $read(write(b, i, x), j) = y\ \wedge a = b\ \wedge i = j$
- Used for:
  - Software verification
  - Hardware verification (memories)

Theories of Interest – Bit-vectors

- 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?

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

Combination of Theories

- In practice, theories are not isolated
- Software verifications needs arithmetic, arrays, bitvectors, …
- Formulas of the following form usually arise:
  - $a = b + 2 \land \mathrm{A} = \mathrm{write}(\mathrm{B}, \mathrm{a} + 1{,}4) \land (read(A, b + 3) = 2 \lor f(a - 1) \neq f(b + 1))$
- The goal of SMT 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.

Eager and Lazy approach of SMT

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

Eager Approach – Example(1)

- 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. $P_{a,b} \wedge P_{b,c} \rightarrow P_{a,c}$

Lazy Approach

- Why "lazy"?
  - Theory information used lazily when checking T-consistency of propositional models

- Characteristics:
  - Modular and flexible
  - Theory information does not guide the search

Lazy Approach - Example

- Consider EUF and the CNF

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

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

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

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? For example:
  - Assume that $a < b,\ b < c$ are in our partial assignment M.
  - If the formula contains $a < c$ we would like to add it to M
- Search guided by T-Solver by finding T-consequences, instead of only validating it as in basic lazy approach.
- Naive implementation:
  - (1) add $\neg l$, (2) if T-inconsistent then infer l
- 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)

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 relatively little effort

DPLL(T)

DPLL(T)

- In a nutshell:
  - DPLL(T) = DPLL(X) + T-Solver
- DPLL(X):
  - Very similar to a SAT solver, enumerates Boolean models
  - Not allowed: pure literal, blocked literal detection, ...
  - 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:
  - $g(a) = c \ \wedge \left(f\big(g(a)\big) \neq f(c) \vee g(a) = d \right) \wedge c \neq d$
  - $1 \qquad \bar{2} \qquad 3 \qquad \bar{4}$
  - $\emptyset \parallel 1,\ \bar{2} \vee 3,\ \bar{4}$ (UnitPropagate)
  - $1 \parallel 1,\ \bar{2} \vee 3,\ \bar{4}$ (UnitPropagate)
  - $1\ \bar{4} \parallel 1,\ \bar{2} \vee 3,\ \bar{4}$ (T-Propagate)
  - $1\ \bar{4}\ 2 \parallel 1,\ \bar{2} \vee 3,\ \bar{4}$ (T-Propagate)
  - $1\ \bar{4}\ 2\ \bar{3} \parallel 1,\ \bar{2} \vee 3,\ \bar{4}$ (Fail)
  - UNSAT

DPLL(T) - Overall algorithm

- High-level 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
}
```

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

Case Reasoning in Theory Solvers (3)

- 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 L has been decided, the T-solver should answer YES/NO)

DPLL(T) - Conflict Analysis

- 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 (unit implication point)
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:= ¬( 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) conjunction of literals $l_1 \wedge \cdots \wedge l_n$ 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 $l_1, \cdots, l_n$ such that:
      - They were in the model when l was T-propagated
      - $l_1 \wedge \cdots \wedge l_n \vDash_T l$
    - Then $reason(l)$ is $\neg l_1 \vee \cdots \vee \neg l_n \vee l$

DPLL(T) - Conflict Analysis (4)

- Let M be of the form … , $c = b$, … and let F contain
  - $h(a) = h(c) \vee p$
  - $a = b \vee \neg p \vee a = d$
  - $a \neq d \vee a = b$

- 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 from $\{h(a) \neq h(c), c = b\}$

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

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 $a \neq d \vee a = b$

$$\dfrac{a = b \vee \neg p \vee \boldsymbol{a = d} \qquad a \neq d \vee a = b}{\boldsymbol{a = b} \vee \neg p}$$

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

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

T-Solver Example: Difference Logic

Difference logic

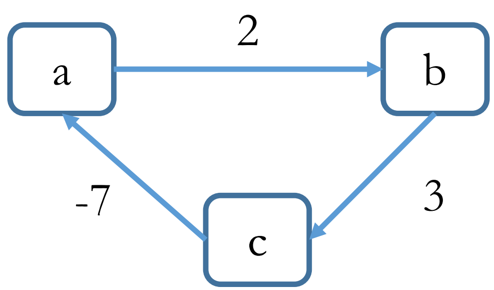
- Literals in Difference Logic are of the form $a - b \bowtie k$, where
  - $\bowtie \in \{\leq, \geq, <, >, =, \neq\}$
  - a and b are integer/real variables
  - k is an integer/real
- At the formula level,
  - $a = b$ is replaced by p and
  - $p \leftrightarrow a \leq b \wedge b \leq a$ is added
- 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
    - e.g. 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 adjacent 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 $G(M)$



- Theorem:
  - M is T-inconsistent iff $G(M)$ has a negative cycle

Difference Logic as a Graph Problem (2)

Theorem:

M is T-inconsistent iff $G(M)$ 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:

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

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 is T-inconsistent iff $G(M)$ has a negative cycle

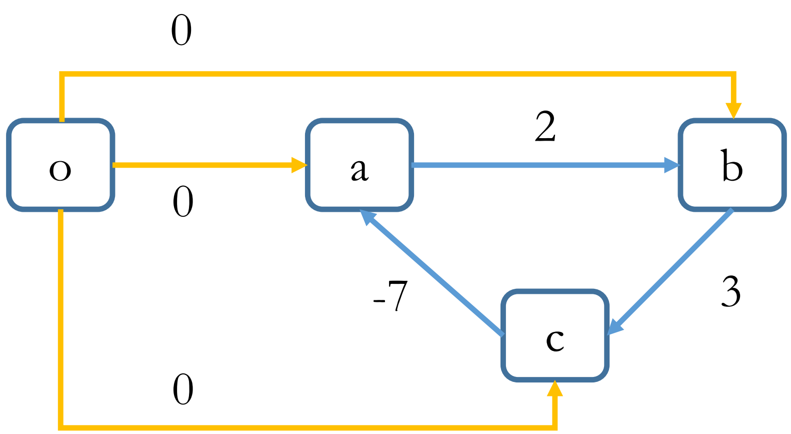
⇒)

Let us assume that there is no negative cycle.

1. Consider additional vertex o with edges $o \xrightarrow{0} v$ to 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 $k = weight(x \longrightarrow y)$!!!

Solution of difference constraints



If $G(M)$ has no negative cycle, then the solution of M is
$\sigma(x) = dist(o, x)$

if $c - a \leq -2$

$$\delta(a) = 0$$
$$\delta(b) = 0$$
$$\delta(c) = -2$$

$$a - b = 0 \leq 2$$
$$b - c = 2 \leq 3$$
$$c - a = -2 \leq -2$$

if $c - a \leq -7$

$$\delta(a) = 0$$
$$\delta(b) = 0$$
$$\delta(c) = -7$$

$$a - b = 0 \leq 2$$
$$b - c = 7 \leq 3$$
$$c - a = -7 \leq -7$$

Bellman-Ford: negative cycle detection

```
forall v ∈ V do d[v] := ∞ endfor
forall i = 1 to |V|-1 do
          forall (u,v) ∈ 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) ∈ 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

- Incrementality easy:
  - Upon arrival of new literal $a \xrightarrow{k} b$ process graph from u

- 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

$$\overbrace{c \longrightarrow^{*} a \xrightarrow{k} b}^{\text{shortest}} \qquad \underbrace{a \xrightarrow{k} b \longrightarrow^{*} d}_{\text{shortest}}$$

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

Theory Combination

Need for Theory Combination

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

$$a = b + 2 \wedge A = write(B, a + 1, 4) \wedge (read(A, b + 3) = 2 \vee f(a - 1) 6 = f(b + 1))$$

- Here reasoning is needed over
  - The theory of linear arithmetic (T_{LA})
  - The theory of arrays (T_{A})
  - The theory of uninterpreted functions (T_{EUF})
- Remember that T-solvers only deal with conjunctions of literals.
- Given T-solvers for the three individual theories, can we combine them to obtain one for ($T_{LA} \cup T_{A} \cup T_{EUF}$) ?

Common Base Theories

Uninterpreted functions QF_UF

$$f(f(x)) = a$$
$$g(a) \neq f(b)$$

Arithmetic

QF_LRA, QF_LIA, ...

$$2x + y \geqslant 3$$
$$x - y > 1$$

Bitvectors

QF_BV

$$\mathtt{bvnot}(x) + 1 = x$$
$$\mathtt{bvuge}(x, 0b000..0)$$

Arrays

QF_AX

$$b = \mathtt{store}(a, i, v)$$
$$x = \mathtt{select}(b, j)$$

- Important: These theories have no non-logical symbol in common (the only thing they share is equality)

Purification

- If F is a formula in theory $T_1 \cup T_2$, we can always transform F into two parts
  - F_1 is in theory T_1
  - F_2 is in theory T_2
- F is satisfiable in $T_1 \cup T_2$ iff $F_1 \land F_2$ is satisfiable (also in $T_1 \cup T_2$)
- This is called purification.
- It's done by introducing new variables to remove mixed terms.

After Purification

- Purification of F produces formulas F_1 in T_1 and F_2 in T_2
- UNSAT Case:
    - If F_1 is unsat in T_1 or F_2 is unsat in T_2 then F is unsat in $T_1 \cup T_2$.
- SAT Case:
    - If F_1 is sat in T_1 and F_2 is sat in T_2 , is F satisfiable in $T_1 \cup T_2$?
    - F_1 has a model M_1 : $M_1 \vDash_{T_1} F_1$
    - F_2 has a model M_2 : $M_2 \vDash_{T_2} F_2$
    - Can we construct a model M such that $M \vDash_{T_1 \cup T_2} F$?

Purification Example

- Formula with mixed terms:

$$x \leq y \land 2y \leq x \land f(h(x) - h(y)) > f(0)$$

- Purification:
  - Separate the uninterpreted function part and the arithmetic part

QF_UF	QF_LRA
$a = h(x)$	$x \leq y$
$b = h(y)$	$2y \leq x$
$d = f(c)$	$c = a - b$
$g = f(e)$	$e = 0$
	$d > g$

Purification Example(2)

- QF_UF part is SAT
  - Possible model with domain = $\{\alpha, \beta\}$

$$
\begin{aligned}
a &= h(x) \\
b &= h(y) \\
d &= f(c) \\
g &= f(e)
\end{aligned}
$$

x	α
y	β
a	α
b	β
c	α
d	β

	α	β
f	β	β
h	α	β

- QF_LRA part is SAT
  - Possible model (with domain = $\mathbb{R}$)

$$
\begin{aligned}
x &\leq y \\
2y &\leq x \\
c &= a - b \\
e &= 0 \\
d &> g
\end{aligned}
$$

x	0
y	0
a	0
b	0

c	0
d	1
e	0
g	0

The two models are not consistent (F is UNSAT)

- One says $x \neq y$, the other says $x = y$
- Their domains have different cardinalities

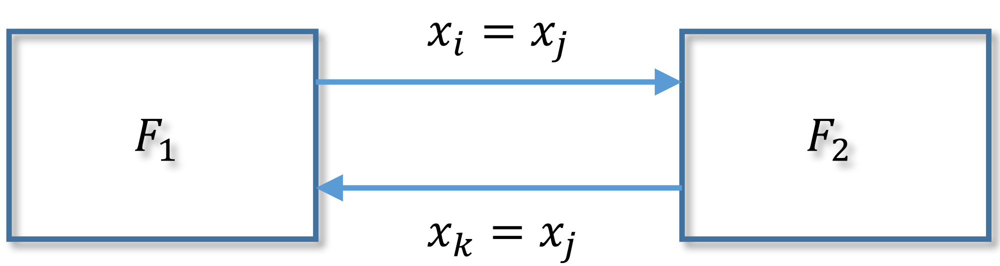
Nelson-Oppen Methond

Central Problem in Theory Combination

- Search for consistent models
  - Start with F in $T_1 \cup T_2$
  - Purify to get F_1 in T_1 and F_2 in T_2
  - Search for two models M_1 and M_2 such that:
    - $M_1 \vDash_{T_1} F_1$ and $M_2 \vDash_{T_2} F_2$
  - M_1 and M_2 have the same cardinality
  - M_1 and M_2 agree on equalities between shared variables
- Nelson-Oppen Method
  - A general framework for solving this problem
  - Originally proposed by Nelson and Oppen, 1979
  - Give sufficient conditions for consistent models to exist
  - Many extensions and variations

The Nelson-Oppen Method (Nelson & Oppen, 1979)

- The theory solvers propagate implied equalities between shared variables.
- If both sides are satisfiable and no-more equalities can be propagated, then F is satisfiable.



Nelson-Oppen Example

QF_UF	QF_LRA
$a = h(x)$	$x \leq y$
$b = h(y)$	$2y \leq x$
$d = f(c)$	$c = a - b$
$g = f(e)$	$e = 0$
	$d > g$

Input formula after purification

Nelson-Oppen Example

QF_UF

$a = h(x)$
$b = h(y)$
$d = f(c)$
$g = f(e)$

$x = y$

QF_LRA

$x \leq y$
$2y \leq x$
$c = a - b$
$e = 0$
$d > g$

$x = y$

QF LRA deduces and propagates $x = y$

Nelson-Oppen Example

QF_UF

$a = h(x)$
$b = h(y)$
$d = f(c)$
$g = f(e)$

$x = y$
$a = b$

QF_LRA

$x \leq y$
$2y \leq x$
$c = a - b$
$e = 0$
$d > g$

$x = y$
$a = b$

QF LRA deduces and propagates $x = y$
QF UF propagates $a = b$

Nelson-Oppen Example

QF_UF

$a = h(x)$
$b = h(y)$
$d = f(c)$
$g = f(e)$

$x = y$
$a = b$
$e = c$

QF_LRA

$x \leq y$
$2y \leq x$
$c = a - b$
$e = 0$
$d > g$

$x = y$
$a = b$
$e = c$

QF LRA deduces and propagates $x = y$
QF UF propagates $a = b$
QF LRA propagates $e = c$

Nelson-Oppen Example

QF_UF

$a = h(x)$
$b = h(y)$
$d = f(c)$
$g = f(e)$

$x = y$
$a = b$
$e = c$
$d = g$

QF_LRA

$x \leq y$
$2y \leq x$
$c = a - b$
$e = 0$
$d > g$

$x = y$
$a = b$
$e = c$
$d = g$

QF LRA deduces and propagates $x = y$
QF UF propagates $a = b$
QF LRA propagates $e = c$
QF UF propagates $d = g$
QF LRA concludes unsat

Nelson-Oppen – Restrictions

- Theories must meet the following restrictions to be decidable in combination:
  - $T_1, \ldots, T_n$ are quantifier-free first-order theories with equality.
  - There is a decision procedure for each of the theories $T_1, \ldots, T_n$.
  - The signatures are disjoint, i.e., for all $1 \leq i < j \leq n$, $\Sigma_i \cap \Sigma_j = \emptyset$.
  - $T_1, \ldots, T_n$ are theories that are interpreted over an infinite domain

Nelson-Oppen –Convex Case

- Deterministic Nelson-Oppen
- Assumptions
  - Given two signature-disjoint, stably-infinite and convex theories T_1 and T_2
  - Given a set of literals S over the signature of $T_1 \cup T_2$

- A theory T is stably-infinite iff every T-satisfiable quantifier-free formula has an infinite model
  - Examples: QF_UF and QF_LRA are stably infinite, QF_BV is not

- A theory T is convex iff

$$S \vDash_T a_1 = b_1 \vee \ldots \vee a_n = b_n$$
$$\Longrightarrow S \vDash a_i = b_i \text{ for some } i$$

Convex Theories

- Definition

  T is convex if, for every set of literals Γ, and every disjunction of variable equalities $x_1 = y_1 \vee \cdots \vee x_n = y_n$, such that

  $$\Gamma \vDash x_1 = y_1 \vee \cdots \vee x_n = y_n ,$$

  we have

  $$\Gamma \vDash x_i = y_i$$

  for some index i.

- Examples
  - QF_UF and QF_LRA are convex
  - QF_LIA, QF_BV, and QF_AX are not convex

Convex Theories - Example

- Linear arithmetic over $\mathbb{R}$ (QF_LRA) is convex

$$x \leq 3 \land x \geq 3 \Rightarrow x = 3$$

- Linear arithmetic over $\mathbb{Z}$ (QF_LIA) is not convex:

while

$$x_1 = 1 \land x_2 = 2 \land 1 \leq x_3 \land x_3 \leq 2 \Rightarrow x_3 = x_1 \lor x_3 = x_2$$

is valid, neither

$$x_1 = 1 \land x_2 = 2 \land 1 \leq x_3 \land x_3 \leq 2 \Rightarrow x_3 = x_1$$

nor

$$x_1 = 1 \land x_2 = 2 \land 1 \leq x_3 \land x_3 \leq 2 \Rightarrow x_3 = x_2$$

is valid.

Non-Convex Theories - Example

- QF_LIA: linear arithmetic over the integers
  $0 \leq x \ \wedge \ x \leq y \ \wedge \ y \leq z \ \wedge \ z \leq 1 \ \vDash \ x = y \ \vee \ y = \ z$

- QF_AX: array theory
  $$b = store(a, i, v) \wedge \ x = select(b, j) \wedge$$
  $$y = select(a, j) \ \vDash \ x = v \ \vee \ x = y$$

Nelson-Oppen – Convex Case

- Given n signature-disjoint, stably-infinite and convex theories $T_1, \ldots, T_n$
  1. Purification: Purify F into $F_1, \ldots, F_n$.
  2. Apply the decision procedure for T_i to F_i. If there exists i such that F_i is unsatisfiable in T_i, return "UNSAT".
  3. Equality propagation: If there exist i, j such that F_i T_i-implies an equality between variables of F that is not T_j-implied by F_j, add this equality to F_j and go to step 2.
  4. Return "SAT"

Example - Convex case

- Consider the following set of literals:

$$f\left(f(x) - f(y)\right) = a$$
$$f(0) = a + 2$$
$$x = y$$

- There are two theories involved: $T_{LA(\mathbb{R})}$ and T_{EUF}
- FIRST STEP:
  - purify each literal so that it belongs to a single theory

Example - Convex case

F: $f\big(f(x) - f(y)\big) = a,\ f(0) = a + 2,\ x = y$

$$f\big(f(x) - f(y)\big) = a$$
$$\Downarrow$$
$$f(e_1) = a$$
$$e_1 = f(x) - f(y)$$
$$\Downarrow$$
$$e_1 = e_2 - e_3$$
$$e_2 = f(x)$$
$$e_3 = f(y)$$

$$f(0) = a + 2$$
$$\Downarrow$$
$$f(e_4) = a + 2$$
$$e_4 = 0$$
$$\Downarrow$$
$$f(e4) = e_5$$
$$e_4 = 0$$
$$e_5 = a + 2$$

Example - Convex case

- SECOND STEP: check satisfiability and exchange entailed equalities

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

- 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

Example - Convex case

- SECOND STEP: check satisfiability and exchange entailed equalities

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

- 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
  - $EUF \models e_2 = e_3$

Example - Convex case

- 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
  - $Ari \vDash e_1 = e_4$

Example - Convex case

- 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
  - $EUF \models a = e_5$

Example - Convex case

- 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 UNSAT
  - Hence the original set of lits was UNSAT

Example – Non-Convex case

- Consider the following set of literals:

  $x \geq 1$
  $x \leq 2$
  $f(x) \neq f(1)$
  $f(x) \neq f(2)$
- There are two theories involved: $T_{LA(\mathbb{Z})}$ and T_{EUF}
- FIRST STEP:
  - purify each literal so that it belongs to a single theory

EUF	Arithmetic	
$f(x) \neq f(a)$	$x \geq 1$	Both theories are SAT …
$f(x) \neq f(b)$	$x \leq 2$	But F is UNSAT
	$a = 1$	
	$b = 2$	

Properties of Nelson-Oppen

- Soundness and Completeness
  - propagating implied equalities is sufficient for some theories but not others
  - the theories for which this is sufficient are called convex theories
  - for these theories, the method is sound and complete

- Termination
  - obvious if the number of shared variables is fixed
  - this is usually the case
  - some theory solvers (e.g., arrays) may dynamically add more variables but this can be bounded

More on Nelson-Oppen

- Can be extended to non-convex theories
  - the theory solvers propagate disjunctions of equalities
- Finding Implied Equalities
  - For QF_UF, decision procedures based on congruence closure give implied equalities for free.
  - It's harder and more expensive for other theories (e.g., linear arithmetic).
  - It gets worse for non-convex theories.
- Delayed Theory Combination
  - Attempt to construct an arrangement lazily in the CDCL(T) framework
  - Create interface equalities and let the SAT solver do the search
  - Different heuristics to decide when and what equalities to create

Nelson-Oppen Method-Non-convex case

Nelson-Oppen – The non-convex case

- Given a formula F that combines n signature-disjoint, stably-infinite theories $T_1, \dots, T_n$
  1. Purification: Purify F into $F_1, \dots, F_n$.
  2. Apply the decision procedure for T_i to F_i. If there exists i such that F_i is unsatisfiable in T_i, return "UNSAT".
  3. Equality propagation: If there exist i, j such that F_i T_i-implies an equality between variables of F that is not T_j-implied by F_j, add this equality to F_j and go to step 2.
  4. Splitting: If there exists i such that
     - $F_i \Rightarrow (x_1 = y_1 \vee \cdots \vee x_k = y_k)$ but $\forall j \in 1, \dots, k.\ F_i \nRightarrow x_j = y_j$,
     - Then apply Nelson-Oppen recursively to: $F \wedge x_1 = y_1, \dots, F \wedge x_k = y_k$
     - If any of these subproblems is satisfiable, return "SAT". Otherwise return "UNSAT"
  5. Return "SAT"

Example – Non-Convex case

- Consider the following set of literals:

  $x \geq 1$
  $x \leq 2$
  $f(x) \neq f(1)$
  $f(x) \neq f(2)$

- There are two theories involved: $T_{LA(\mathbb{Z})}$ and T_{EUF}
- FIRST STEP:
  - purify each literal so that it belongs to a single theory

EUF	Arithmetic	
$f\ (x) \neq f(a)$	$x \geq 1$	Both theories are SAT …
$f\ (x) \neq f(b)$	$x \leq 2$	But F is UNSAT
	$a = 1$	
	$b = 2$	

Example – Non-Convex case

EUF

$f\ (x) \neq f(a)$

$f\ (x) \neq f(b)$

Arithmetic

x ≥ 1

x ≤ 2

a = 1

b = 2

Case separation:

$(x = a) \lor (x = b)$

EUF

$f\ (x) \neq f(a)$

$f\ (x) \neq f(b)$

$x = a$

UNSAT

Arithmetic

x ≥ 1

x ≤ 2

a = 1

b = 2

$x = a$

EUF

$f\ (x) \neq f(a)$

$f\ (x) \neq f(b)$

$x = b$

UNSAT

Arithmetic

x ≥ 1

x ≤ 2

a = 1

b = 2

$x = b$

Example – Non-convex case

- Consider the following UNSATISFIABLE set of literals:

$$1 \le x \le 2$$
$$f(1) = a$$
$$f(x) = b$$
$$a = b + 2$$
$$f(2) = f(1) + 3$$

- There are two theories involved: $T_{LA(\mathbb{Z})}$ and T_{EUF}
- FIRST STEP:
  - purify each literal so that it belongs to a single theory

Example – Non-convex case

- F:

$$1 \leq x \leq 2$$
$$f(1) = a$$
$$f(x) = b$$
$$a = b + 2$$
$$f(2) = f(1) + 3$$

$$f(1) = a$$
$$\Downarrow$$
$$f(e_1) = a$$
$$e_1 = 1$$

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

Example – Non-convex case

- 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
  - $EUF \vDash a = e4$

Example – Non-convex case

- 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...

Example – Non-convex case

- 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.

Example – Non-convex case

- SECOND STEP: check satisfiability and exchange entailed equalities

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

- 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
  - $EUF \vDash_T a = b$, that when sent to Ari makes it UNSAT

Example – Non-convex case

- SECOND STEP: check satisfiability and exchange entailed equalities

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

- Let's try now with $x = e_2$
  - Ari-Solver says SAT
  - EUF-Solver says SAT
  - $EUF \models_T b = e_3$, that when sent to Ari makes it UNSAT

Example – Non-convex case(7)

- 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

Non-Deterministic Nelson-Oppen (Tinelli & Harandi, 1996)

- Assumptions
  - Two theories T_1 and T_2 that share no non-logical symbol and are stably infinite
  - F is a conjunction of literals of $T_1 \cup T_2$
  - F is purified to F_1 in T_1 and F_2 in T_2

- Stably Infinite Theories
  - A theory T is stably infinite if every formula that's satisfiable in T has an infinite model
  - Examples: QF_UF and QF_LRA are stably infinite, QF_BV is not

Variable Arrangements

- Definition
  - Let V be the set of all variables that are shared by F_1 and F_2
  - An arrangement of V is a conjunction of variable equalities and disequalities that define a partition of V
- Example
  - If $V = \{x_0, x_1, x_2, x_3\}$ and we partition V into three subsets $\{x_0, x_1\}$, $\{x_2\}$, and $\{x_3\}$ then the corresponding arrangement is

$$x_0 = x_1 \wedge x_0 \neq x_2 \wedge x_1 \neq x_2 \wedge$$
$$x_0 \neq x_3 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3$$

Non-Deterministic Nelson-Oppen (continued)

- Procedure
  - Guess a partition of the variables V and let $\mathcal{A}$ be the corresponding arrangement
  - Check whether $F_1 \wedge \mathcal{A}$ is satisfiable in T_1 and $F_2 \wedge \mathcal{A}$ is satisfiable in T_2
- Theorem
  - If $F_1 \wedge \mathcal{A}$ is satisfiable in T_1 and $F_2 \wedge \mathcal{A}$ is satisfiable in T_2 then F is satisfiable in $T_1 \cup T_2$.
- Why this works (informally)
  - T_1 and T_2 are stably infinite. This implies that they have models of the same infinite cardinality.
  - The arrangement$\mathcal{A}$ forces the two models to agree on equalities between shared variables.

Non-Deterministic Nelson-Oppen (continued)

- Issues
  - How do we find the right arrangement?
    - The number of possible partitions of a set of n variables is known as Bell's number (B_n)
    - This grows very fast with n (e.g., B_{11} is 27644437)
    - We can't possibly try them all
  - How do we handle theories that are not stably infinite?

Model-Based Theory Combination

Model-Based Theory Combination

- Models are available
  - The theory solvers for T_1 and T_2 produce models when F_1 and F_2 are SAT:
  $$M_1 \vDash_{T_1} F_1 \text{ and } M_2 \vDash_{T_2} F_2$$
  - The Nelson-Oppen methods do not use these models
- Model-based theory combination: Make use of the models M_1 and M_2 :
  - if M_1 and M_2 are consistent, done
  - optionally, attempt to modify M_1 and M_2 to make them consistent
  - if that fails, add constraints to cause CDCL(T) to backtrack and search for other models

Combining a Theory with QF_UF

- Very Common Case
  - One theory is QF_UF and the other is either an arithmetic theory or QF_BV
- QF_UF has good properties
  - Deciding satisfiability is cheap (fast congruence closure algorithms)
  - These algorithms give the implied equalities for free
  - It's stably infinite
- Model-Based Combination With QF_UF
  - Works with an arbitrary theory T (non-convex, non-stably infinite)
  - Main components:
    - congruence closure
    - interface lemmas
    - model mutation and reconciliation

Congruence Closure

- Key problem in QF_UF

  Given a finite set of terms and some equalities between them

  $$t_1 = u_1 , \dots , t_m = u_m$$

  find all the implied equalities

- Congruence Closure Algorithms

  Construct an equivalence relation $\sim$ between terms such that

  if $t_i = u_i$ is an original equality then $t_i \sim u_i$

  $\sim$ is closed under the congruence rule:

  $$v_1 \sim w_1 , \dots , v_k \sim w_k \Rightarrow f(v_1 , \dots , v_k) \sim f(w_1 , \dots , w_k)$$

  The $\sim$ relation contains all the implied equalities:

  $$t_1 = u_1 , \dots , t_n = u_n \Rightarrow t = u \text{ iff } t \sim u$$

Congruence Closure Example

- Terms: $a, b,\ f(a),\ f(f(a)),\ f(f(f(a)),\ f(b)$
- Initial Equalities: $f(f(a)) = a, f(a) = b$
- Equivalence Relation
  - Initially
    - $\{ a, f(f(a))\}\ \{b, f(a)\}\ \{f(b)\}\ \{f(f(f(a))\ \}$
  - Congruence: $f(a) = f(f(f(a))$
    - $\{a, f(f(a))\}\ \{b, f(a), f(f(f(a)))\}\ \{f(b)\}$
  - Congruence: $f(b) = f(f(a))$
    - $\{a, f\left(f(a)\right), f(b)\}\ \left\{b, f(a), f\left(f\left(f(a)\right)\right)\right\}$
  - Done

Checking Satisifiability in QF_UF

- A QF_UF formula can be written as a conjunction of equalities and disequalities:
  $(t_1 = u_1 \wedge \cdots \wedge t_n = u_n) \wedge (v_1 \neq w_1 \wedge \cdots \wedge v_m \neq w_m)$
- To check satisfiability
  - compute the congruence closure $\sim$ of the equalities
  - if $v_i \sim w_i$ for some i then return UNSAT else return SAT

- Example
  - Formula: $f\big(f(a)\big) = a \ \wedge \ f(a) = b \ \wedge \ b \neq f(f(f(a))$
  - Congruence closure: $\{a, f(f(a)), f(b)\}\ \{b, f(a), f(f(f(a)))\}$
  - So the formula is UNSAT

Building Models in QF_UF

- From a Congruence Closure
  - Basic idea: one element in the domain per equivalence class in the congruence closure
  - We can always ensure that every term t is interpreted as its class representative
- Example
  - Formula: $f(b) = a \wedge b = f(a) \wedge a \neq f(c)$
  - Congruence closure: $\{a, f(b)\}\ \{b, f(a)\}\ \{c\}\ \{f(c))\}$
  - Model:
    - domain = $\{\alpha, \beta, \gamma, \delta\}$

a	α
b	β
c	γ

	α	β	γ	δ
f	β	α	δ	α

Flexibility in QF UF Models

- Enlarging the domain
  - Let F be a satisfiable QF_UF formula and M a model of F
  - For any cardinal $k > |M|$, we can construct a new model M' of cardinality k that satisfies F
  - This implies that QF_UF is stably infinite
- Shrinking the domain
  - We can sometimes make the domain smaller by modifying the congruence closure
  - Previous example:
    - F is $f(b) = a \wedge b = f(a) \wedge a \neq f(c)$
    - Congruence closure: $\{a, f(b)\}$ $\{b, f(a)\}$ $\{c\}$ $\{f(c)\}$
  - We could merge $\{f(c)\}$ and $\{b, f(a)\}$ to get a new relation $\sim'$: $\{a, f(b)\}$ $\{b, f(a), f(c)\}$ $\{c\}$
  - A model built from $\sim'$ still satisfies F

Basic Model-Based Combination With QF_UF

- Assumptions
  - A formula F in $QF\_UF \cup T$
  - After purification: F_1 in QF_UF and F_2 in T
  - V denotes the set of variables shared by F_1 and F_2
  - $\sim$ is the equivalence relation computed by congruence closure from F_1
- Procedure
  - If F_1 is not satisfiable, return UNSAT
  - Get all equalities implied by F_1
  - Let H be the set of implied equalities that are between variables of V
  - Check whether $F_2 \wedge H$ is satisfiable in T; if not return UNSAT
  - Otherwise, get a model M for $F_2 \wedge H$.
  - If M does not conflict with relation $\sim$ return SAT
  - Otherwise, add interface lemmas to force backtracking

Basic Model-Based Combination With QF_UF - Conflicts

- Conflicts
  - M conflicts with E if there are two shared variables x and y such that
    $$M \vDash x = y \text{ but } x \nsim y$$
  - conflicts in the other direction are not possible (since $M \vDash H$)
- If there are no conflicts
  - M and $\sim$ agree on equalities between shared variables
  - We can extend M by adding an interpretation for all the uninterpreted functions in the QF_UF part
  - We get a new model M' that satisfies F_2 and F_1

Interface Lemmas

- Interface lemma for x and y
  - A formula that encodes "$x = y$ in T" ⇒ "$x = y$ in QF_UF"
  - The exact formulation depends on the implementation and theory involved

- Examples
  - T is QF_LRA: we add the clause $x = y \lor x > y \lor y > x$
  - T is QF_BV: we add the clause $\neg(bveq\ \ x\ y) \lor x = y$
  - in these clauses, ($x = y$) must be an atom handled by the QF_UF solver

- If M conflicts with $\sim$ on $x = y$, this lemma forces the SMT solver to backtrack and search for different models

Imrovements

- Model Mutation
  - Exploit flexibility in the Simplex-based arithmetic solver.
  - There may be many solutions to a set of linear arithmetic constraints.
  - Mutation: modify the Simplex model to give distinct values to distinct interface variables.
  - This reduces the risk of accidental conflicts

Improvements (continued)

- Model Reconciliation
  - Exploit flexibility in QF_UF to eliminate conflicts while keeping M fixed
  - If x and y are in conflict: $M \vDash x = y$ and $x \nsim y$
  - To try to resolve this conflict:
    - tentatively merge the equivalence classes of x and y
    - propagate the consequences by congruence closure
    - accept the merge unless if makes the QF_UF part UNSAT or it would propagate new equalities to theory T