

An Introduction to SMT Solvers

Johannes Kanig

INRIA, LRI, ProVal team

2 juin 2010

AdaCore

Contents

Introduction

Overview

SMT Solvers

- Equality Reasoning

- Arithmetic

- Combination of Theories

- Satisfiability

- Combining SAT and the Theories

- Matching

Looking Elsewhere

Conclusion

Automated Theorem Proving

Who does prove Theorems today?

- ▶ Mathematicians
- ▶ Hardware verification
- ▶ Software verification

Automation of proofs

- ▶ Makes life easier for mathematicians
- ▶ Makes it possible to state (and prove) thousands of tiny theorems
- ▶ Makes large-scale verification of hardware and software accessible and more reliable

The ProVal Team

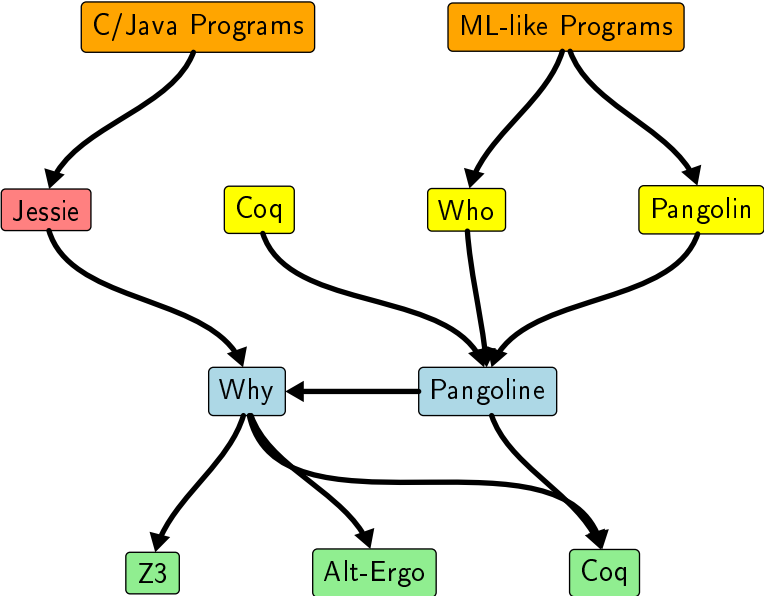
Main Topic: Program proof

- ▶ Specification languages
- ▶ Computing proof obligations (formulas that imply the correctness of the program)
- ▶ Proving proof obligations

Tools (Why Platform)

- ▶ **Caduceus**, **Krakatoa**: C/Java frontends
- ▶ **Jessie**: Common intermediate language
- ▶ **Why**: Obtain proof obligations from programs
- ▶ **Alt-Ergo**: Automated prover (SMT Solver)

The Why platform



A logical formula ...

$sorted(t, i, j) =$

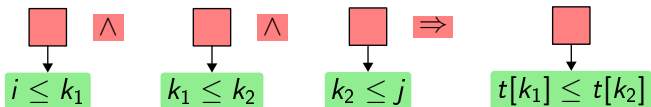
$$\forall k_1, k_2 : int . i \leq k_1 \wedge k_1 \leq k_2 \wedge k_2 \leq j \Rightarrow t[k_1] \leq t[k_2]$$


... as seen by an SMT solver


$sorted(t, i, j) =$


$\forall k_1, k_2 : int$

\Downarrow



 Instantiation

 Logic reasoning

 Theory reasoning (here: Arithmetic)

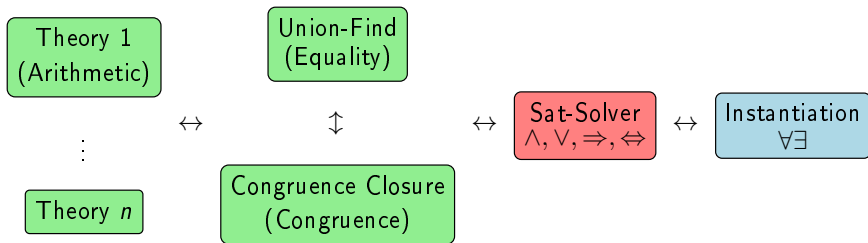
Satisfiability Modulo Theories

SMT provers divide the problem in three parts

- ▶ The **theory** part: equality reasoning, arithmetic reasoning, ...
- ▶ The **satisfiability** part: deals with logical connectors
 \wedge , \vee , \Rightarrow , \neg , ...
- ▶ The **instantiation** of quantified axioms

We will look at each of the three parts in turn

The different parts of an SMT solver



A more detailed example

Hypotheses

- ▶ $H_1 : a > 0$
- ▶ $H_2 : \forall xy. x \geq y \rightarrow \max(x, y) = x$

Goal

$$G : f(\max(a, 0)) = f(a)$$

Solved by an SMT Solver (1)

Negate the Goal

$H_1 \wedge H_2 \rightarrow G$ becomes $H_1 \wedge H_2 \wedge \neg G$

Launch Sat-Solver

Assume H_1 , H_2 and $\neg G$ and try to derive a contradiction

- ▶ Assume the inequality $a > 0$
- ▶ Register the **lemma**: $\forall xy. x \geq y \rightarrow \max(x, y) = x$
- ▶ Assume the inequality $f(\max(a, 0)) \neq f(a)$
- ▶ Currently no contradiction!

Instantiation

Specialize the lemma by applying it to a and 0 and replace \rightarrow :
 $a \geq 0 \rightarrow \max(a, 0) = a \quad \Leftrightarrow \quad a < 0 \vee \max(a, 0) = a$

Solved by an SMT Solver (2)

Split the disjunction

First assume $a < 0$, then assume $\neg(a < 0)$, try to find a contradiction in both cases

Assuming $a < 0$

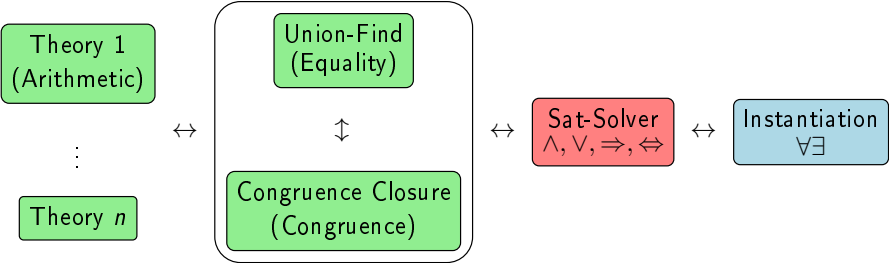
Direct contradiction with H_1 (using knowledge about the symbols $<$ and \geq)

Assuming $\neg(a < 0)$

- ▶ It follows $\max(a, 0) = a$
- ▶ Deduce $f(\max(a, 0)) = f(a)$
- ▶ Contradiction with $\neg G$

We have obtained a contradiction in all cases, the negated formula is unsatisfiable, that means the input formula is valid!

Equality Reasoning



Equality reasoning - The problem

Terms

$t ::= c \mid f(t_1, \dots, t_n)$

Given

a list of equations $t = t'$

We want to know

Does the equation $t_1 \stackrel{?}{=} t_2$ follow?

Using the axioms

Reflexivity $t = t$

Symmetry $t_1 = t_2 \rightarrow t_2 = t_1$

Transitivity $t_1 = t_2 \wedge t_2 = t_3 \rightarrow t_1 = t_3$

Congruence $t_1 = t_2 \rightarrow f(t_1) = f(t_2)$

Example

Given

- ▶ $f^2(a) = f(f(a)) = a$
- ▶ $f^5(a) = f(f(f(f(f(a)))))) = a$

We want to prove

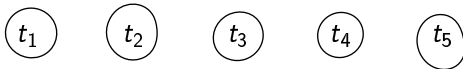
$$f(a) = a$$

Proof

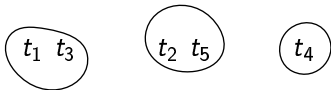
1. $f^5(a) = f^3(a)$ (Congruence)
2. $f^2(a) = f^3(a) = a$ (Transitivity, Symmetry)
3. $f^3(a) = f(f^2(a)) = f(a)$ (Congruence)
4. $f(a) = a$ (Transitivity of (2) and (3))

Disjoint Sets

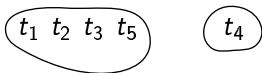
- ▶ Goal: deal with the first **three** axioms efficiently
- ▶ Idea: put all terms into disjoint sets
- ▶ When two terms are in the same set, they are equal
- ▶ Initial state: every term is in his own set:



- ▶ After treating $t_1 = t_3$ and $t_2 = t_5$:



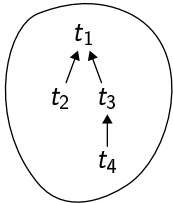
- ▶ After treating $t_1 = t_2$:



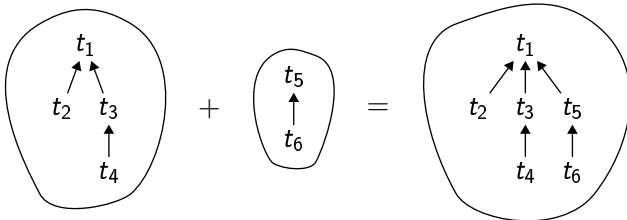
- ▶ Deciding $t \stackrel{?}{=} t'$ amounts to checking if t, t' are in the same set

Union-Find (1975)

- ▶ Represent each set by a tree with upward pointers:



- ▶ The root is the **representative**
- ▶ Operation **find** to find the representative of any term: just follow the arrows
- ▶ Operation **union** to treat an equality: simply point one root to the other



Two important optimizations

- ▶ Keep trees small: let point root of smaller tree to root of larger tree
- ▶ **Path compression**: “flatten” trees, each time we are searching for a root r starting from t , let t point directly to r afterwards
- ▶ Result: Algorithm is quasi-linear (optimal)
- ▶ **Incrementality**: we can add equations one by one, interleave equations $t_1 = t_2$ with queries $t_1 \stackrel{?}{=} t_2$

Inequalities $t_1 \neq t_2$

- ▶ Simply maintain the information that two sets of terms must be different
- ▶ Merging sets for which an inequality was registered leads to an inconsistency

Congruence Closure (1980)

- Deal with the fourth axiom: Congruence

$$\forall xy. x = y \rightarrow f(x) = f(y)$$

for any function symbol f

- Solution: represent a term by a directed acyclic graph (DAG) with sharing. Example: $f(f(a, b), b)$



- Add an equivalence relation to this graph (using union-find):



represents $f(f(a, b), b) = a$

Finding new equalities

- ▶ Build a reverse dictionary mapping nodes to their fathers:

$$a \mapsto f(a, b), g(a)$$

$$b \mapsto f(a, b)$$

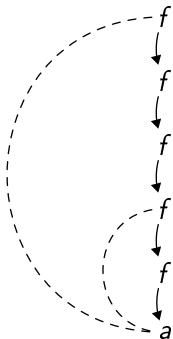
- ▶ Two new operations: **find** and **merge**.

```
merge(t1, t2) =  
  union(t1, t2);  
  F1, F2 = fathers(t1), fathers(t2);  
  for each x in F1, y in F2 do  
    if congruent(x, y) then merge(x, y);  
  done
```

Congruence Closure — Example

Given

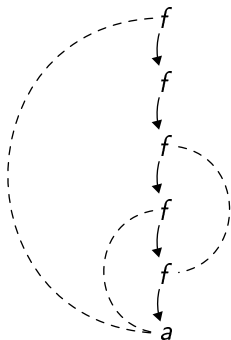
- ▶ $f^2(a) = f(f(a)) = a$
- ▶ $f^5(a) = f(f(f(f(f(a)))))) = a$



Congruence Closure — Example

Given

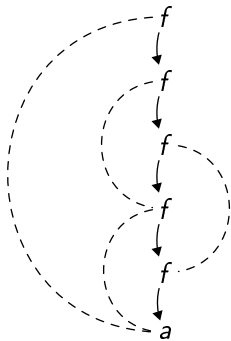
- ▶ $f^2(a) = f(f(a)) = a$
- ▶ $f^5(a) = f(f(f(f(f(a)))))) = a$



Congruence Closure — Example

Given

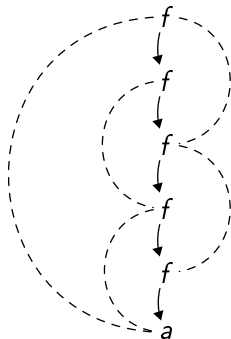
- ▶ $f^2(a) = f(f(a)) = a$
- ▶ $f^5(a) = f(f(f(f(f(a)))))) = a$



Congruence Closure — Example

Given

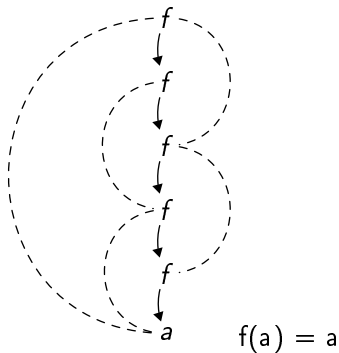
- ▶ $f^2(a) = f(f(a)) = a$
- ▶ $f^5(a) = f(f(f(f(f(a)))))) = a$



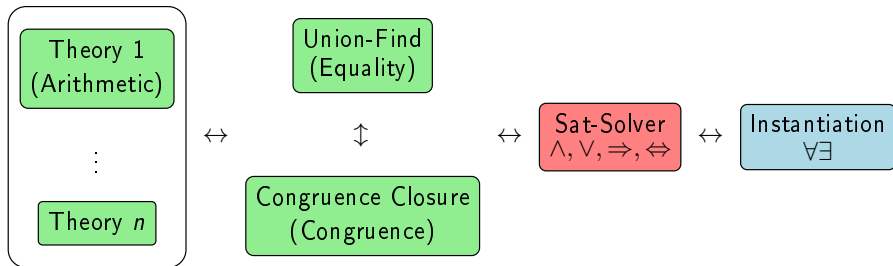
Congruence Closure — Example

Given

- ▶ $f^2(a) = f(f(a)) = a$
- ▶ $f^5(a) = f(f(f(f(f(a))))) = a$



Theory Reasoning (Arithmetic)



Arithmetic reasoning

Arithmetic

- ▶ **Interprets** the function symbols $+$, $-$, \times , \div , and the arithmetic constants
- ▶ But also the relation symbols \leq , $<$, \geq , $>$

There are a few algorithms to deal with Linear Arithmetic

- ▶ Gauss Elimination (Equality only)
- ▶ Fourier-Motzkin
- ▶ Simplex Algorithm

We will look more closely at these methods

Gauss Elimination

Goal: deal with **equalities** in linear arithmetics

- ▶ Transform term into sums of monomials: $\sum_i^k c_i t_i$
- ▶ When treating an equality between such polynomes

$$\sum_i^k c_i t_i = \sum_j^k d_j s_j$$

isolate a monomial, say, t_1 , and build the equation

$$t_1 = \sum_j^k \frac{d_j}{c_1} s_j - \sum_{i \neq 1}^k \frac{c_i}{c_1} t_i$$

Fourier-Motzkin Algorithm (1)

Goal: deal with **inequalities** in linear arithmetics

basic notions

- ▶ An inequality C in canonical form:

$$\sum_{i=1}^n a_i x_i \leq a_0 \quad a_i \in \mathbb{Q}$$

- ▶ Note αC the multiplication of an inequation with a coefficient α :

$$\sum_{i=1}^n \alpha a_i x_i \leq \alpha a_0$$

- ▶ Note $C_1 + C_2$ the addition of two inequations :

$$\sum_{i=1}^n (a_i + b_i) x_i \leq a_0 + b_0$$

Fourier-Motzkin Algorithm (2)

Set $I = \{C_1 \cdots C_n\}$ the starting set of inequations. Each step of the algorithm will eliminate a variable from the set of the equations.

- ▶ Let I^+ (I^-) be the set of equations where x appears with positive (negative) coefficient
- ▶ Compute

$$I_x = \bigcup_{C \in I^-, D \in I^+} \beta C + \alpha D \quad \alpha x \in C, -\beta x \in D$$

- ▶ Let I_0 the set of inequations in I without x
- ▶ Replace I par $I' = I_0 \cup I_x$
- ▶ In particular, if x appears only with coefficients of the same sign in I , **suppress** all inequations where x appears
- ▶ When I does not contain variables any more, either we have satisfiable inequalities (like $1 \leq 2$) or an inconsistency

Fourier-Motzkin Algorithm (3)

- ▶ Complexity: double exponential
- ▶ Not incremental
- ▶ Still behaves well in practice
- ▶ Can be easily extended to deduce equations between terms

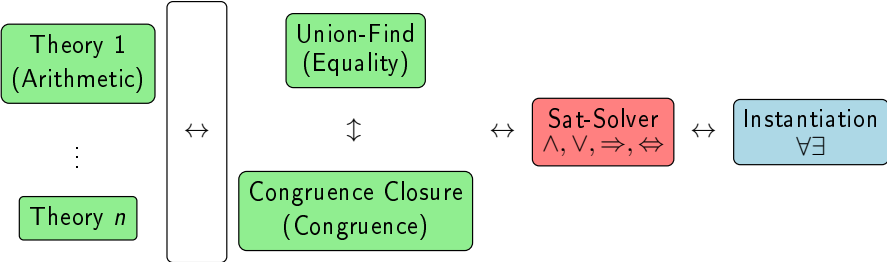
Simplex Algorithm (1947)

- ▶ Initially designed to solve the optimization problem:

$$\begin{array}{ll} \text{maximize} & \vec{c} \cdot \vec{x} \\ \text{subject to} & A\vec{x} \leq \vec{b} \quad \vec{x} \geq \vec{0} \end{array}$$

- ▶ Complexity: exponential (polynomial in practice)
- ▶ incremental variants exist

Combination of Theories



Combination of Theories

We have seen different **decision procedures** for different theories:

- ▶ The theory of **uninterpreted function symbols** (Congruence Closure)
- ▶ Linear arithmetics (Fourier-Motzkin, Simplex)

Other decision procedures exist:

- ▶ For the theory of pairs (or lists): `nil` and `cons`
- ▶ The theory of arrays: `get` and `set`
- ▶ The theory of bitvectors: `get`, `append`, `shift`

How do they work together?

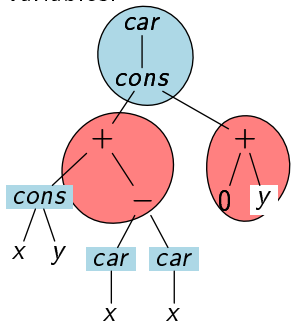
Nelson Oppen Combination — 1980 (1)

Consider a term with symbols from multiple theories:

$$\text{car}(\text{cons}(\text{cons}(x, y) + (\text{car}(x) - \text{car}(x)), y + 0))$$

Variable abstraction

Introduce variables to build **pure** terms and equations between variables:



$$z_1 = \text{car}(x)$$

$$z_2 = \text{cons}(x, y)$$

$$z_3 = z_2 + (z_1 - z_1)$$

$$z_4 = y + 0$$

$$z_5 = \text{car}(\text{cons}(z_3, z_4))$$

Nelson Oppen Combination (2)

First step

Transform the problem into n **pure** problems and a set of equations between auxiliary variables, using variable abstraction

Second step

- ▶ Run each decision procedure independently on its pure equations
- ▶ Communicate newly discovered equalities to the other decision procedures

Shostak Combination — 1978 (1)

Characteristics

- ▶ avoids costly equality propagation in Nelson-Oppen
- ▶ Requires a **solver** instead of a decision procedure: a solver takes an equation $t_1 = t_2$ and returns a substitution $x_i \mapsto t_i$
- ▶ Requires a **canonizer**, putting terms of the theory in normal form (sum of monomials, ...)
- ▶ Restriction: applies only to theories of **equality** (inequalities in linear arithmetic not well supported)

Shostak Combination (2)

- ▶ Maintain a table mapping terms to their representatives (for example using an E-Graph with Union-find)

Terms	Representatives
t_1	r_1
t_2	r_1
t_3	r_2

- ▶ When treating an equation $a = b$, call the solver on a and b , obtain a substitution σ and apply this substitution to all representatives. Finally, canonize all representatives.

Shostak vs. Nelson-Oppen

Comparison

- ▶ Shostak supposedly faster (is it in practice?)
- ▶ It's harder to write a solver than it is to write a decision procedure
- ▶ Shostak works well in the case of Congruence Closure + other theory
- ▶ It is still active research to combine more than just one theory with CC in the Shostak framework
- ▶ Most solvers use a Nelson-Oppen like Approach
- ▶ Alt-Ergo uses a modified Shostak-like Approach

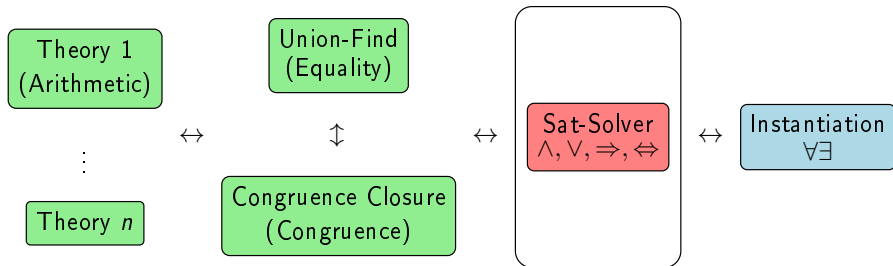
Theories in SMT solvers: Summary

- ▶ SMT solvers use Congruence Closure and various decision procedures for the “Theory part” of SMT
- ▶ SMT solvers are generally **complete** and terminating for conjunctions of atoms in this fragment of the theory
- ▶ If they don't prove the goal, it's wrong!

Beware

- ▶ non-linear arithmetic is undecidable
- ▶ on integer arithmetics, many provers use incomplete (but faster) heuristics

Satisfiability



Satisfiability

We have seen . . .

- ▶ How to decide if a **conjunction** of equalities and inequalities is inconsistent

But what about the other connectors?

- ▶ Negation, disjunction, implication, equivalence

This is the work of a SAT-solver

A new input language

Atoms

- ▶ Formulas whose structure a SAT-solver will not look at
- ▶ Typically, equalities / inequalities, predicates
- ▶ Examples: $x = 0$, $x \leq f(x) + 3$, $even(t)$

Literal

An atom, with or without negation sign \neg

Conjunctive Normal form (CNF)

- ▶ A formula in CNF is a conjunction of disjunctions:

$$(a \vee b \vee \neg b) \wedge (b \vee c) \wedge (\neg c \vee d \vee e)$$

- ▶ The disjunctions are called **clauses**
- ▶ This is the input language of a SAT solver

Sat Solvers

Sat Solvers

- ▶ Decide if a formula in CNF is **satisfiable**
- ▶ In a satisfiable formula one can assign a truth value to each atom so that the formula becomes true
- ▶ Such an assignment is called a **model**
- ▶ This problem is **NP-complete**

Putting formulas in CNF

We won't talk about it here, we just mention. . .

- ▶ That the naive way of doing it leads to a formula that is exponentially bigger than the original one
- ▶ There are smarter ways of doing it that produce a CNF linear in the size of the original formula, but the formula is still much bigger
- ▶ One can do the conversion lazily, during the execution of the SAT algorithm (makes the algorithm more complicated)

DPLL (1)

The DPLL method

- ▶ Invented by Davis, Putnam (1960) and Davis, Logemann, Loveland(1962)
- ▶ Is an intelligent exhaustive search of a model for the input formula

A Unit Clause

- ▶ Is a clause which consists of a single literal
- ▶ This literal must be true if the formula is satisfiable

DPLL (2)

Boolean Constraint Propagation (BCP)

- ▶ Search for a unit clause and assume the corresponding literal l to be true
- ▶ Each literal of the form $\neg l$ is removed
- ▶ Each clause which contains l (positively) is removed
- ▶ Repeat

Case splitting

- ▶ Choose an arbitrary literal l occurring in the formula
- ▶ Try to find a model where l is true
- ▶ If there is none, try to find a model where $\neg l$ is true (backtracking)
- ▶ If there is none either, the formula is unsatisfiable

DPLL (3)

The algorithm

- ▶ As long as there are unit clauses, do BCP (simplify current formula)
- ▶ If there are no unit clauses, do a case split
- ▶ if the current formula contains the empty clause, it is unsatisfiable; backtrack

The algorithm as inference rules

$$\text{Axiom } \frac{}{\Gamma \vdash \Delta, \emptyset}$$

$$\text{Unit } \frac{\Gamma, l \vdash \Delta}{\Gamma \vdash \Delta, l}$$

$$\text{Elim } \frac{\Gamma, l \vdash \Delta}{\Gamma, l \vdash \Delta, l \vee C}$$

$$\text{Reduce } \frac{\Gamma \vdash \Delta, C}{\Gamma, l \vdash \Delta, \neg l \vee C}$$

$$\text{Split } \frac{\Gamma, l \vdash \Delta \quad \Gamma, \neg l \vdash \Delta}{\Gamma \vdash \Delta}$$

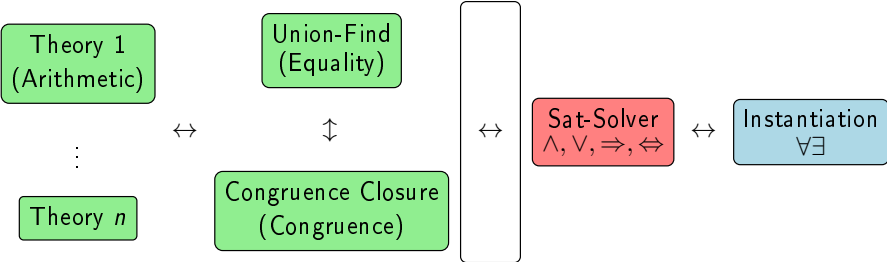
DPLL(4)

- ▶ DPLL is correct and complete
- ▶ Choice of the splitting variable is very important
- ▶ There are many heuristics to choose it

Check for the validity of a formula using DPLL

- ▶ Negate the input formula
- ▶ Search for a model of this formula (DPLL)
- ▶ If there is none, the formula is valid
- ▶ Otherwise the discovered model is a **counterexample**

Combining SAT and the theories



SAT and the Theory working together

The General Idea

- ▶ SAT decomposes the logical structure of the formula
- ▶ Each assumed literal is given to the appropriate theory
- ▶ which one? consider $t.[i - 1] = x + y$
- ▶ A conflict can be detected in two ways:
 - ▶ The SAT: We have found the empty clause
 - ▶ The Theory: We have derived an absurd statement in some (combination of) theories

$$x = x + 1$$

DPLL — Backjumping

Optimisation

- ▶ Reconsider the Split rule:

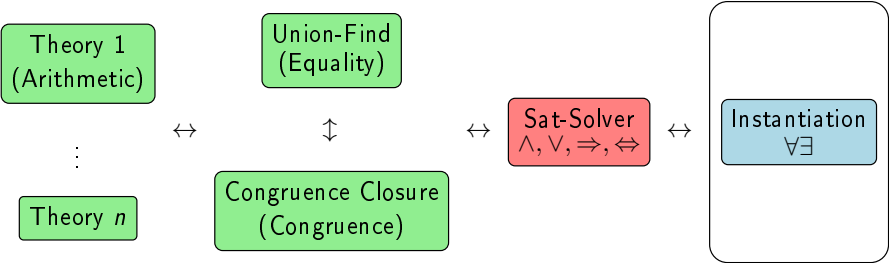
$$\text{Split} \frac{\Gamma, I \vdash \Delta \quad \Gamma, \neg I \vdash \Delta}{\Gamma \vdash \Delta}$$

- ▶ Backtracking to $\neg I$ not always needed!
- ▶ When I is not involved in the inconsistency in the left branch, then one does not need to check the branch of $\neg I$

Collaboration needed

- ▶ The **SAT** must be able to explain conflicts
- ▶ The **Theory** must be able to explain conflicts

Instantiation



Dealing with quantifiers

Example

Axiom A : $\forall x : int. f(g(x), 0) = 0$

- ▶ Potentially, for any term t of integer type, add $f(g(t), 0) = 0$ to the knowledge base
- ▶ Where do the terms t come from?
- ▶ Usually, the nodes of the E-graph are considered (the set of **ground terms**)
- ▶ This is incomplete (a proof may require to invent a new term)
- ▶ Problem: taking all terms of integer type to instantiate axiom A is too much

Triggers (1)

Restricting the set of terms to consider

- ▶ Idea: do not allow instantiation of all terms of the right type
- ▶ Add a **trigger**, a term of a certain form to the axiom:
- ▶ Axiom $A : \forall x : \text{int}[g(x)]. f(g(x), 0) = 0$
- ▶ Instantiate x only with terms t for which ground term $g(t)$ exists
- ▶ This is an heuristics (even more incomplete)
- ▶ Triggers can be given by the user, or computed by the prover

Triggers (2)

Matching

- ▶ Given a term t with variables and a ground term t' , try to find a **substitution** σ for the variables in t such that $t\sigma = t'$
- ▶ This substitution is used to instantiate the entire axiom

Considerations

- ▶ A trigger can consist of several terms
- ▶ The trigger term(s) should contain all variables of the axiom
- ▶ The process of comparing the ground terms with the trigger is called **matching**
- ▶ A large trigger is more restrictive than a small trigger: We instantiate less (which is good), but we may miss necessary instantiations
- ▶ A variable as trigger is as good (as bad) as no trigger

E-Matching

Use the E-Graph to enhance matching

- ▶ Use the equality information of the E-graph to match more terms
- ▶ Example: The E-graph of $f(a) = a$ actually describes also $f(f(a))$ and $f(f(f(a))) \dots$



- ▶ A trigger of the form $f(a)$ can be instantiated by any $f^n(a)$

Skolemization

What to do with existential quantifiers?

- ▶ We can eliminate them using **Skolemization**
- ▶ An axiom $\exists x.P(x)$ becomes $P(c)$ for a fresh **Skolem** constant c
- ▶ If quantifiers are nested:

$$\forall x \exists y.P(x, y)$$

becomes

$$\forall x.P(x, f(x))$$

for a fresh **Skolem function** f

The Big Picture (1)

- ▶ Take a first-order formula as input
- ▶ Negate it — try to prove that the negation is unsatisfiable
- ▶ Eliminate Existentials (Skolemization)
- ▶ Compute triggers (if not given) for quantifiers
- ▶ Put it in CNF (or do it lazily)
- ▶ Run the SAT solver on the CNF
- ▶ Every assumed atom is given to the theories
- ▶ Inconsistencies can be detected by
 - ▶ the SAT: an empty clause present
 - ▶ the theory: set of assumed literals is inconsistent wrt. the theory
- ▶ At some point (when? how often?), do **Matching** to instantiate axioms, obtain new terms and new facts

The Big Picture (2)

- ▶ SMT provers are generally **complete** over the SAT + theory part (with restrictions)
- ▶ SMT provers are **incomplete** over the matching/instantiation part

SMTLib, SMTComp

- ▶ There is a common input format accepted by most SMT solvers: SMTLib
- ▶ SMT provers: Alt-Ergo, Z3, Yices, CVC3 (all with quantifiers and theories), MathSat, Boolector (without quantifiers)
- ▶ There is an annual competition with many different categories: with/without quantifiers, arithmetics, bit-vectors, arrays, . . .
- ▶ 2009 winner: mostly Yices (in categories without quantifiers) and CVC3 (in categories with quantifiers)
- ▶ in a few categories, 2008 winner Z3 was actually faster (but not officially part of 2009 competition)

Resolution based provers (1)

Unification

Generalization of **Matching**: Given two terms t_1 and t_2 **both** containing variables, find a substitution σ such that $t_1\sigma = t_2\sigma$.

Resolution rule

$$\text{Resolution} \frac{C_1 \vee l_1 \quad C_2 \vee \neg l_2 \quad \sigma = \text{unif}(l_1, l_2)}{C_1\sigma \vee C_2\sigma}$$

Example

$$\text{Resolution} \frac{\neg P(x) \vee Q(x) \quad P(a) \quad \sigma = [x \mapsto a]}{Q(a)}$$

Resolution based provers (2)

Algorithm

- ▶ Just as SMT solvers, negate formula, skolemize and put in CNF
- ▶ Now, using the Resolution rule, derive new clauses
- ▶ When encountering the empty clause, the initial formula was valid

TPTP Competition

- ▶ Participants: Vampire, Spass, E, ...
- ▶ Recent winners of various categories: Vampire, Waldmeister

Resolution based provers (3) - Comparison

Advantages

- ▶ Resolution is **complete** for first-order logic
- ▶ It implements a **semi**-decision procedure:
 - ▶ If the formula is valid, the algorithm eventually stops, with the right answer
 - ▶ If the formula is invalid, it may run forever
- ▶ Resolution is much better at proving complex theorems **without arithmetic reasoning**

Disadvantages

- ▶ Integrating equality reasoning is difficult, but has been mastered: Superposition, Paramodulation
- ▶ Integration of arithmetic or other theories is currently out of reach of the state of the art

Tableau based provers (Zenon)

Tableau method

- ▶ Do **not** transform formula into CNF, but do negate it
- ▶ Apply rules to formula to build a refutation tree
- ▶ Rules for conjunction and disjunction:



- ▶ If a branch contains a contradiction (A and $\neg A$, $x \neq x$, False, ...), it is **closed**
- ▶ When all branches are closed, the negated formula is unsatisfiable
- ▶ Prominent prover in this class: Zenon
- ▶ It is the only prover that produces a machine-checkable proof
- ▶ Tableau method generally considered to be slower than

Conclusion

SMT provers . . .

- ▶ Divide the input formula into a theory part, a propositional part, and lemmas
- ▶ Are **complete** wrt. the propositional part, and selected theories
- ▶ Are **incomplete** in general
- ▶ The combination of decision procedures, but also the combination of the SAT solver and the theories are the main challenges of SMT