Program verification

Nelma Moreira

Departamento de Ciência de Computadores da FCUP

Decidable first order theories and SMT Solvers
Lecture 10

SMT Solvers

- SAT can codify operations and relations between integers with bounded precision
  - using representations as bit vectors
  - representing addition, etc as Boolean circuits
- As well as, other finite datatypes and structures
- But, cannot represent unbounded types (e.g., reals) or infinity data structures (stacks, lists)
- Bounded arithmetic is not very efficient for large values
- There are efficient procedures for these FOL theories for conjunctions of atomic formulas
- Use search strategies based on SAT solvers
- Are called SMT (*Satisfiability Modulo Theories*) solvers.

- Infinity set of variables $x_1, x_2, \ldots$ (*Vars*)
- Logic symbols: Boolean connectives ($\wedge$, $\vee$, $\implies$, $\neg, \ldots$),quantifiers ($\forall$, $\exists$) and parentheses '(', ')'.
- Non-logic symbols: (ranked) alphabet Σ for functional ($f, g, \ldots$) and predicate symbols ($R, P, Q, \ldots$)
- Syntax: for terms and for formulae

$$t ::= x \mid a \mid f(t, \ldots, t)$$

$$\varphi ::= R(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \implies \varphi) \mid \forall x\varphi \mid \exists x\varphi$$

- A formula is *closed* if does not have free variables.
- Semantics: a Σ structure A with domain A, interpretation of non logic symbols ($\cdot^A$) and a assignment of domain elements to the free variables, $\alpha : \mathit{Vars} \to A$. The assigmenet extends to terms.
- A formula φ is satisfiable if there exists a structure A of Σ and assignment α such that φ is true ($A \models_\alpha \varphi$)
- A formula is valid if for all structures A of Σ, φ is true ($\models \varphi$)

(i) $A \models_\alpha t_1 = t_2$ if $\alpha(t_1) = \alpha(t_2)$

(ii) $A \models_\alpha R(t_1, \ldots, t_n)$ if $(\alpha(t_1), \ldots, \alpha(t_n)) \in R^{\mathcal{A}}$

(iii) $A \models_\alpha \neg\varphi$ if $A \not\models_\alpha \varphi$

(iv) $A \models_\alpha \varphi \wedge \psi$ if $A \models_\alpha \varphi$ and $A \models_\alpha \psi$

(v) $A \models_\alpha \varphi \vee \psi$ if $A \models_\alpha \varphi$ or $A \models_\alpha \psi$

(vi) $A \models_\alpha \varphi \implies \psi$ if $A \not\models_\alpha \varphi$ or $A \models_\alpha \psi$

(vii) $A \models_\alpha \forall x\, \varphi$ if for all $a \in A$ if $A \models_{\alpha[a/x]} \varphi$ where:

$$\alpha[a/x](y) = \begin{cases} \alpha(y) & \text{if } y \neq x \\ a & \text{if } y = x \end{cases}$$

(viii) $A \models_\alpha \exists x\, \varphi$ if exists $a \in A$ such that $A \models_{\alpha[a/x]} \varphi$

Given formulae φ and ψ, we have the following decision problems

Validity problem: Is φ valid? ($\models \varphi$?)

Satisfiability problem: Is φ satisfiable? (exist A, α, $A \models_\alpha \varphi$)

Consequence problem: Is ψ a consequence of φ? ($\psi \models \varphi$? or $\Gamma \models \varphi$ for Γ a set of formulae)

Equivalence problem: Are ψ and φ equivalent?

These are, in some sense, variations of the same problem :

$\models \varphi \iff \neg\varphi$ is unsatisfiable
$\psi \models \varphi \iff \neg(\psi \implies \varphi)$ is unsatisfiable
$\psi \equiv \varphi \iff (\psi \models \varphi \wedge \varphi \models \psi)$
φ is satisfiable $\iff \neg\varphi$ is not valid

A solution to a decision problem is a program that takes problem instances as input and always terminates, producing a correct *yes* or *no* output.

A decision problem is decidable if it has a solution. A decision problem is undecidable if it is not decidable.

Theorem (Church & Turing)

- *The decision problem of validity in first-order logic is undecidable: no program exists which, given any φ, decides whether $\models \varphi$.*
- *The decision problem of satisfiability in first-order logic is undecidable: no program exists which, given any φ, decides whether φ is satisfiable.*

However, there is a procedure that halts and says yes if is valid.

A decision problem is semi-decidable if exists a procedure that, given an input,

- halts and answers "yes" $\iff$ "yes" is the correct answer,
- halts and answers "no" if "no" is the correct answer,
- or does not halt if "no" is the correct answer

Unlike a decidable problem, the procedure is only guaranteed to halt if the correct answer is "yes".

The decision problem of validity in first-order logic is semi-decidable.

Let Σ be an alphabet of a first-order language.

- A theory $\mathcal{T}$ is a set of of closed formulae such that $\mathcal{T} \models \varphi$ implies $\varphi \in \mathcal{T}$ (closed under derivability).
- A $\mathcal{T}$-structure is a Σ -structure that validates all formulae of $\mathcal{T}$
- A formula φ is $\mathcal{T}$-satisfiable if it is satisfiable in a $\mathcal{T}$-structure; in the same way we define $\mathcal{T}$-valid
- A theory $\mathcal{T}$ is finitely (recursively) axiomatizable if there exists a finite (recursive) set $\mathcal{A} \subseteq \mathcal{T}$ (axioms) such that $\forall \varphi, \varphi \in \mathcal{T} \iff \mathcal{A} \models \varphi$
- A theory $\mathcal{T}$ is complete if for all closed formulae φ, $\mathcal{T} \models \varphi$ or $\mathcal{T} \models \neg\varphi$.
- A theory $\mathcal{T}$ is decidable if for all closed formulae it is possible to decide if $\mathcal{T} \models \varphi$.
- A axiomatizable and complete theory is decidable.

- Given a Σ structure A,
$$Th(A) = \{\varphi \mid A \models \varphi\}$$
is complete.
- Semantically defined theories are important as they allow to reason about mathematical domains (naturals, integers, algebraic structures, etc.), but they must be axiomatizable.
- A fragment of a theory $\mathcal{T}$ is a subset of formulae of $\mathcal{T}$ with a syntactic restriction, e.g.:
  - only conjunctions of literals;
  - quantifier-free;
  - etc.

1. Equality Theory and Uninterpreted Functions $\mathcal{T}_E$
2. Theory of Arithmetic - Peano Axioms $\mathcal{T}_{PA}$
3. Theory of Arithmetic of Presburger (additive fragment) $\mathcal{T}_{\mathbb{N}}$
4. Linear Integer Arithmetic $\mathcal{T}_{\mathbb{Z}}$ (same expressiveness of $\mathcal{T}_{\mathbb{N}}$)
5. Real Arithmetic and Linear Rational Arithmetic ($\mathcal{T}_{\mathbb{R}}$, $\mathcal{T}_{\mathbb{Q}}$)
6. Set Theory of Zermelo-Frankle
7. Geometry Theory (Euclides, non-standard, etc.)
8. Group Theory
9. Theory of Regular Languages (expressions) (Kleene Algebras)

$$\forall x.x = x$$
$$\forall x, y.x = y \rightarrow y = x$$
$$\forall x, y, z.x = y \wedge y = z \rightarrow x = z$$
$$\forall \bar{x}, \bar{y}.x_1 = y_1 \wedge \cdots x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$
$$\forall \bar{x}, \bar{y}.x_1 = y_1 \wedge \cdots x_n = y_n \rightarrow P(x_1, \ldots, x_n) \rightarrow P(y_1, \ldots, y_n)$$

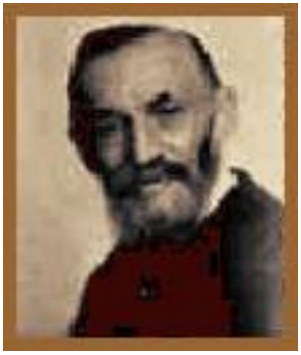
The last two axioms are congruences: functional and predicates.

$\mathcal{T}_E$-validity is undecidable. Quantifier-free fragment is decidable. Fragment with only functions where the conjunctive fragment is $\mathcal{EUF}$, is decidable.

First Incompleteness Theorem Kurt Gödel (1931)

Any effectively generated (i.e., recursively enumerable) theory capable of expressing elementary arithmetic cannot be both consistent and complete. In particular,for any consistent, effectively generated formal theory that proves certain basic arithmetic truths, there is an arithmetical statement that is true, but not provable in the theory.

A semantic theory Th(M), where M interprets each symbol with its standard mathematical meaning in the interpretation domain, is always a complete theory. Therefore, the semantic theories of natural numbers and integers cannot be axiomatizable, not even by an infinite recursive set of axioms.



Let $\Sigma = \{0, 1, +, x, =, <\}$. The axioms define basic facts of naturals and $+$ and x ($\mathbb{N} \models \mathcal{PA}$):

1. $\forall x(x + 1 \neq 0)$
2. $\forall x \forall y(x + 1 = y + 1 \rightarrow x = y)$
3. $0 + 1 = 1$
4. $\forall x \; x + 0 = x$
5. $\forall x \forall y \; x + (y + 1) = (x + y) + 1$

Let $\Sigma = \{0, 1, +, x, =, <\}$. The axioms define basic facts of naturals and $+$ and x ($\mathbb{N} \models \mathcal{PA}$):

1. $\forall x(x + 1 \neq 0)$
2. $\forall x \forall y(x + 1 = y + 1 \rightarrow x = y)$
3. $0 + 1 = 1$
4. $\forall x \; x + 0 = x$
5. $\forall x \forall y \; x + (y + 1) = (x + y) + 1$
6. $\forall x \; x \times 0 = 0$
7. $\forall x \forall y \; x \times (y + 1) = (x \times y) + x$
8. (induction principle) $(Q(0) \; \wedge \; (\forall x(Q(x) \rightarrow Q(x + 1)) \rightarrow \forall x Q(x)$

$T_{\mathcal{PA}}$-validity is undecidable (Gödel's Incompleteness). Even the quantifier-free fragment of $\mathcal{T}_{PA}$ is undecidable. (Matiyasevich, 1970).

- If we exclude axioms 6 and 7 we obtain Presburger Arithmetics $\mathcal{T}_{\mathbb{N}}$ which is complete and decidable.
- This theory has many applications in formal verification and is related with automata theory.
- Linear integer arithmetic, $\mathcal{T}_{\mathbb{Z}}$, $(\Sigma = \{\ldots, -1, 1, 0, 1, 2, \ldots +, =, <\})$ reduces to Presburger theory.

- If we exclude axioms 6 and 7 we obtain Presburger Arithmetics $\mathcal{T}_{\mathbb{N}}$ which is complete and decidable.
- This theory has many applications in formal verification and is related with automata theory.
- Linear integer arithmetic, $\mathcal{T}_{\mathbb{Z}}$, $(\Sigma = \{\ldots, -1, 1, 0, 1, 2, \ldots +, =, <\})$ reduces to Presburger theory.
- Suppose

$$\forall w, x. \exists y, z. x + 2y - z - 13 > -3w + 5.$$

we can introduce new variables v_p and v_n (in $\mathbb{N}$) for each variable v

$$\forall w_p, w_n, x_p, x_n. \exists y_p, y_n, z_p, z_n. (x_p - x_n) + 2(y_p - y_n) - (z_p - z_n) - 13 > -3(w_p - w_n) + 5,$$

change the side of the $-$ to

$$\forall w_p, w_n, x_p, x_n. \exists y_p, y_n, z_p, z_n. x_p + 2y_p + z_n + 3w_p > x_n + 2y_n + z_p + 13 + 3w_n + 5$$

and code in unary.

- Of course $\mathcal{T}_{\mathbb{N}}$ reduces to $\mathcal{T}_{\mathbb{Z}}$:
- The $\mathcal{T}_{\mathbb{N}}$-formula

$$\forall x \exists y . x = y + 1$$

is equisatisfiable to the $\mathcal{T}_{\mathbb{Z}}$-formula

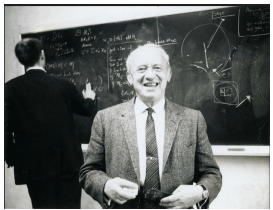
$$\forall x . (x > -1 \implies \exists y . y > -1 \wedge x = y + 1)$$

Let $\Sigma = \{0, 1, +, x, =, \geq\}$. The Real Arithmetic (or elementary algebra) is :

- with addition an abelian group $(\mathbb{R}, +, 0)$, $+$ associative and commutative, 0 identity and all elements have inverse $(-)$.:
- with multiplication a ring $(\mathbb{R}, +, \times, 1, 0)$: $\times$ associative and distributes over addition, 1 identity.
- and a field: $\times$ commutative, $1 \neq 0$, non 0 elements have multiplicative inverse.
- closed: $\geq$ total order and

1. $\forall x, y, z.\ x \geq y \implies x + z \geq y + z$
2. $\forall x, y.\ x \geq 0 \wedge y \geq 0 \implies xy \geq 0$
3. $\forall x. \exists y.\ x = y^2 \vee x = -y^2$
4. for each odd integer n , polynomials of odd degree have at least one root.

$$\forall \vec{x}. \exists y. y^n + x_1 y^{n-1} + \cdots + x_{n-1} y + x_n = 0$$

Tarski  proved in the 1930s that $\mathcal{T}_{\mathbb{R}}$ is decidable, although the Second World War prevented his publishing the result until 1956. Collins (1975) proposed a more efficient technique of cylindrical algebraic decomposition (CAD) CAD runs in time proportionate to $2^{2^{k|F|}}$, for some constant k and for $|F|$ the length of F..

The full theory of rational numbers (with addition and multiplication) is undecidable, since the property of being a natural number can be encoded in it. For the linear theory of rationals the alphabet is $\Sigma = \{0, 1, +, =, \geq\}$ and corresponds to $\mathcal{T}_{\mathbb{R}}$ without multiplication.

1. $\forall x, y.\ x \geq y \wedge y \geq x \implies x = y$
2. $\forall x, y, z.\ x \geq y \wedge y \geq z \implies x \geq z$
3. $\forall x, y. x \geq y \vee y \geq x$
4. $\forall x, y, z.(x + y) + z = x + (y + z)$
5. $\forall x. x + 0 = x$
6. $\forall x. x + (-x) = 0$
7. $\forall x, y. x + y = y + x$
8. $\forall x, y, z. x \geq y \implies x + z \geq y + z$
9. for each positive integer n, $\forall x. nx = 0 \implies x = 0$
10. for each positive integer n, $\forall x. \exists y. x = ny$

Models are divisible torsion-free abelian groups. (Axiom 9).

- Difference logic is a fragment (a sub-theory) of linear arithmetic.
- Atomic formulas have the form $x - y \leq c$, for variables x and y and constant c.
- Conjunctions of difference arithmetic inequalities can be checked very efficiently for satisfiability by searching for negative cycles in weighted directed graphs.
- Graph representation: each variable corresponds to a node, and an inequality of the form $x - y \leq c$ corresponds to an edge from y to x with weight c.
- The quantifier-free satisfiability problem is solvable in $O(|V||E|)$.

The alphabet is $\Sigma_L = \{cons, head, tail, atom, =\}$.

$$\mathcal{T}_E$$
$$\forall x, y.\ head(cons(x, y)) = x$$
$$\forall x, y.\ tail(cons(x, y)) = y$$
$$\forall y.\ \neg atom(y) \implies cons(head(y), tail(y)) = y$$
$$\forall x, y \neg atom(cons(x, y))$$

- $atom(x)$ is a predicate that is true if the argument x is a singleton.
- *cons* is a constructor
- In Lisp *head* is *car* (contents of address register) and *cdr* (contents of decrement register).
- The axioms of $\mathcal{T}_E$ ensure that *head* and *tail* are functional congruences and *atom* a predicate congruence.
- Satisfiabitily of the quantifier-free fragment is decidable.
- Can be extended to other recursive data structures $\mathcal{T}_{RDS}$:binary trees, stacks, etc.

The alphabet is $\Sigma_A = \{read, write, =\}$. Arrays are functions that can be modified. The term $read(a, i)$ corresponds to $a[i]$, and $write(a, i, v)$ corresponds to $a[i \leftarrow v]$.

$$\mathcal{T}_E$$

$$\forall a, i, j.\ i = j \rightarrow read(a, i) = read(a, j)$$

$$\forall a, i, j, v.\ i = j \rightarrow read(write(a, i, v), j) = v$$

$$\forall a, i, j, v.\ \neg(i = j) \rightarrow read(write(a, i, v), j) = read(a, j)$$

$$\forall a, b.\ (\forall i.\ read(a, i) = read(b, i)) \rightarrow a = b\ (extensionality)$$

$\mathcal{T}_A^=$-validity is undecidable. Quantifier-free fragment is decidable. Without the last axiom (extensionality) ($\mathcal{T}_A$) even that fragment was not decidable.

- *Fixed-size bit-vectors*
  - Model bit-level operations of machine words, including 2^n-modular operations (where n is the word size), shift operations, etc.
  - Decision procedures for the theory of fixed-size bit vectors often rely on appropriate encodings in propositional logic.
- *Pointer logic*, allows to reasoning about variables that refer to some other program construct, such as a variable, a procedure or an address. A pointer corresponds to the unique address of a memory cell. The way the memory cells are addresses is given by the memory model. It is characterised by
  - *memory valuation* $M : A \longrightarrow D$ where A is a set of addresses and D the set of data words stored in each memory cell; and
  - a *memory layout* $L : V \longrightarrow A$ that associates to each program variable an address.

- Are specific to a given theory.
- Determine if a formula is inconsistent, satisfiable, or valid.
- Can work on conjunctions of atomic formulae or decide if a formula is consequence of other formulae.
- Can use heuristics to improve performance, but have to give the correct answer and terminate (sound and complete).

- As we saw there are many useful decidable theories (or at least fragments):
  - Equality with uninterpreted functional symbols $\mathcal{EUF}$

$$x = y \wedge f(f(f(x))) = f(x) \rightarrow f(f(f(f(f(y))))) = f(x)$$

  - Updates of functions, registers and tuples
  - Linear integer and rational arithmetics and ($\mathcal{LIA}, \mathcal{LRA}$, linear programming, Simplex, etc.)

$$x \leq y \wedge x \leq 1 - y \wedge 2x \geq 1 \rightarrow 4x = 2$$

  - Difference logic

$$x - y < c$$

  - Bit vectors: modular aritmetics
  - Lists and other RDS.
  - Pointer Logics
- In general, combinations of decidable theories are also decidable

Decidable theories

Theory	**Complexity**
PL	NP-complete
$T_{\mathbb{N}}$, $T_{\mathbb{Z}}$	$\Omega\left(2^{2^n}\right)$, $O\left(2^{2^{2^{kn}}}\right)$
$T_{\mathbb{R}}$	$O\left(2^{2^{kn}}\right)$
$T_{\mathbb{Q}}$	$\Omega\left(2^n\right)$, $O\left(2^{2^{kn}}\right)$
T^{+}_{RDS}	not elementary recursive

Complexities for quantifier-free, conjunctive fragments of theories

Theory	**Complexity**	**Theory**	**Complexity**
PL	$\Theta(n)$	T_{E}	$O(n \log n)$
$T_{\mathbb{N}}$, $T_{\mathbb{Z}}$	NP-complete	$T_{\mathbb{R}}$	$O\left(2^{2^{kn}}\right)$
$T_{\mathbb{Q}}$	PTIME	T^{+}_{RDS}	$\Theta(n)$
T_{RDS}	$O(n \log n)$	T_{A}	NP-complete

Let

$$x + 2 = y \implies f(read(write(a, x, 3), y - 2) = f(y - x + 1)$$

What theories are involved here?

- equality and uninterpreted functions, $\mathcal{T}_E$
- arrays, $\mathcal{T}_A^=$
- arithmetic, $\mathcal{T}_{\mathbb{Z}}$

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ two theories with alphabets Σ_1 and Σ_2; andaxioms A_1 and A_2.

The combined theory $\mathcal{T}_1 \cup \mathcal{T}_2$ such that $\Sigma_1 \cap \Sigma_2 = \{=\}$ is given by:

- alphabet $\Sigma_1 \cup \Sigma_2$
- axioms: $A_1 \cup A_2$

Nelson & Oppen, 1979

Satisfiability of the quantifier-free fragment of $\mathcal{T}_1 \cup \mathcal{T}_2$ is decidable if

- satisfiability of the quantifier-free fragment of $\mathcal{T}_1$ is decidable
- satisfiability of the quantifier-free fragment of $\mathcal{T}_2$ is decidable
- and certain technical requirements are met

- Extend SAT solvers to FOL
- Use decision procedures alone or combined to decide conjunctions of atomic formulae
- SMT use the propositional backbone of the formulae
  - Use search strategies of modern SAT solvers
  - Terms are substituted by propositional variables
  - Find a solution with a SAT solver
  - If found, consider the interpretation of the variables and evaluates the FOL formula with the appropriate solver.

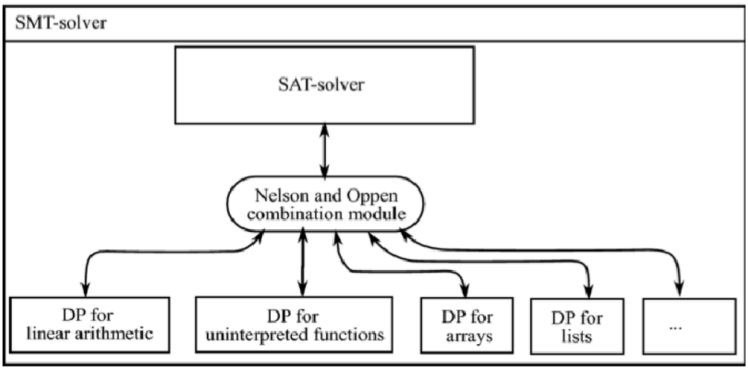
Let prop(φ) be a function that maps $\varphi \in \mathcal{T}$ (in CNF and quantifier-free) into a propositional formula (substituting atomic formulae by propositional variables) and unprop the inverse function. Given an assignment ρ for prop(φ) let

$$\varphi(\rho) = \{\text{unprop}(p_i) \mid \rho(p_i) = \top\} \cup \{\neg\text{unprop}(p_i) \mid \rho(p_i) = \bot\}$$

```
SMT–Solver(φ) {
    A := prop(φ)
    loop
        (r, ρ) := SAT(A)
        if r= unsat then return unsat
        (θ, r):=DP_T(φ(ρ))
        if r=sat then return sat
        C := ⋁_{B∈θ} ¬prop(B)
        A := A ∧ C
}
```

where $\theta \subseteq \varphi(\rho)$ corresponds to unsatisfiable formulae. Then we add to A the propositional equivalent of θ (C) to ensure that the assignment ρ is not used again in SAT. DP_T is the decision procedure for $\mathcal{T}$.

Basic architecture

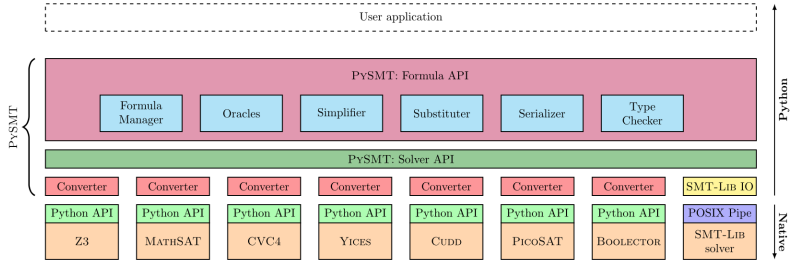


- In the last two decades, SMT procedures have undergone dramatic progress. There has been enormous improvements in efficiency and expressiveness of SMT procedures for the more commonly occurring theories.
- The annual competition for SMT procedures plays an important rule in driving progress in this area.
- A key ingredient is SMT-LIB, an online resource that proposes, as a standard, a unified notation and a collection of benchmarks for performance evaluation and comparison of tools.
- Some SMT solvers: Z3, CVC4, Alt-Ergo, Yices 2, MathSAT, Boolector, etc.
- Usually, SMT solvers accept input either in a proprietary format or in SMT-LIB format.

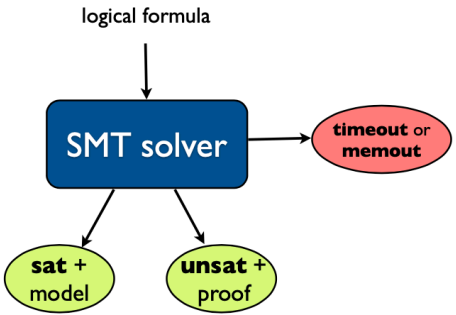
- SMT-LIB: The Satisfiability Modulo Theories Library
  `http://smtlib.cs.uiowa.edu`
- SMT-COMP: The Satisfiability Modulo Theories Competition
  `https://github.com/SMT-COMP`
  `http://www.smtcomp.org`
- Decision procedures - an algorithmic point of view
  `https://www.decision-procedures.org/`
- SAT Association `http://satassociation.org/sat-smt-school.html`
- SAT/SMT Examples `https://sat-smt.codes`

- Catalog of theory declarations - semi-formal specification of theories of interest
  - A theory defines a vocabulary of sorts and functions. The meaning of the theory symbols are specified in the theory declaration.
- Catalog of logic declarations - semi-formal specification of fragments of (combinations of) theories
  - A logic consists of one or more theories, together with some restrictions on the kinds of expressions that may be used within that logic.
- Library of benchmarks
- Utility tools (parsers, converters, ...)
- Useful links (documentation, solvers, ...)
- SMT-LIB language expresses logical statements in a many-sorted first-order logic.

- The pySMT library allows a Python program to communicate with several SMT solvers based on a common language.
- This makes it possible to code a problem independently of the SMT solver, and run the same problem with several SMT solvers.



φ valid $\iff$ $\neg\varphi$ unsatisfiable



If a solver cannot find a solution perhaps other can.

The aim is to solve combinations such as

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_2 = x - 4) \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$
$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leq 1) \wedge (x_2 \geq 3)$$
$$(i = j \wedge a[j] = 1) \wedge \neg(a[i] = 1)$$

We consider quantifier-free theories, T, for which there exists a decision algorithm DP_T for the conjunction of atomic formulae.

- Corresponds to the equality theory $\mathcal{T}_E$ only with variables (and constants that can be eliminated) and quantifer-free

$$\varphi := \varphi \wedge \varphi \mid (\varphi) \mid \neg\varphi \mid t = t$$
$$t := x|c$$

- has the same expressivity and complexity of propositional logic.

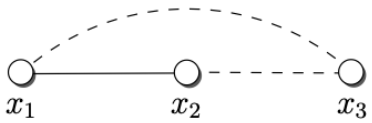
Exerc.

Describe an algorithm to eliminate constants from a formula with equalities. $\diamond$

Decision procedure for theory of equality (conjunctions), DP_T

- Seja φ a conjunction of equalities and inequalities
- Build a graph $G = (N, E_=, E_\neq)$ where
- N are variables of φ,
- $E_=$, edges (x_i, x_j) correspond to equalities $x_i = x_j \in \varphi$ (dashes)
- $E_\neq$, edges (x_i, x_j) correspond to inequalities $x_i \neq x_j \in \varphi$ (filled)
- φ is not satisfiable if and only if there exists an edge $(v_1, v_2) \in E_\neq$ such that v_2 is reachable from v_1 by edges of $E_=$.

For $x_2 = x_3 \wedge x_1 = x_3 \wedge x_1 \neq x_2$, we conclude that is not satisfiable



There are two approaches for the Boolean combination of atomic formulas

- *eager*
  - translate to an equisatisfiable propositional formula
  - that is solved by a SAT solver
- *lazy*
  - incrementally encode the formula in a proposicional formula
  - use DPLL SAT solver
  - use a solver for the theory (DP_T) to refine the formula and guide the SAT solver
- the lazy approach seems to work better

Mainly in the case that φ contains other connectives besides conjunction is better to integrate D_T in a SAT solver.

- Suppose φ in (NNF)
- $at(\varphi)$ set of atomic formulae over Σ in φ; $at_i(\varphi)$ i-th atomic formula
- To each atomic formula $a \in at(\varphi)$ associate $e(a)$ a propositional variable, called the *encoder*
- Extend the encoding e to φ, and let $e(\varphi)$ be the formula resulting from substituting each Σ-literal by its encoder.
- For example if $\varphi := (x = y \vee x = z)$ then $e(\varphi) := e(x = y) \vee e(x = z)$

Let

$$\varphi := x = y \wedge ((y = z \wedge \neg(x = z)) \vee x = z)$$

We have

$$e(\varphi) := e(x = y) \wedge ((e(y = z) \wedge \neg(e(x = z))) \vee e(x = z)) := \mathcal{B}$$

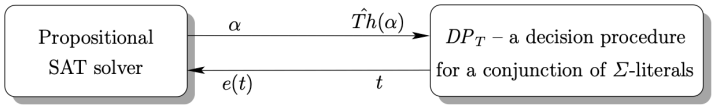
Using a SAT solver we obtain an assignment for $\mathcal{B}$:

$$\alpha := \{e(x = y) \mapsto \text{true}, e(y = z) \mapsto \text{true}, e(x = z) \mapsto \text{false}\}$$

The procedure DP_T checks if the conjunction of literals correspondent to α is satisfiable, i.e.,

$$\hat{Th}(\alpha) = (x = y) \wedge (y = z) \wedge x \neq z$$

This formula is not satisfiable, thus $\neg\hat{Th}(\alpha)$ is a tautology. We can make the conjunction $e(\neg\hat{Th}(\alpha)) \wedge \mathcal{B}$ and call again the SAT solver but α will be blocked as it will not satisfy $e(\neg\hat{Th}(\alpha))$ (*blocking clause*).



Let α' be a new assignment

$$\alpha' := \{e(x = y) \to \mathsf{true}, e(y = z) \to \mathsf{true}, e(x = z) \to \mathsf{true}\}$$

that corresponds to

$$\hat{Th}(\alpha') := (x = y) \wedge (y = z) \wedge x = z$$

which is satisfiable, proving that the original formula φ is satisfiable.

If one considers a partial assinment such that:

$$\alpha' := \{e(x = y) \rightarrow \text{true}, e(y = z) \rightarrow \text{true}$$

Then DP_T could infere that $x = z$ holds and inform the SAT solver of $e(x = z) \rightarrow$ true and $e(x \neq z) \rightarrow$ false which could correspond to BCP.

Formally, given a encoding $e(\varphi)$ and an assignment α, for each encoder $e(at_i)$ we have

$$Th(at_i, \alpha) = \begin{cases} at_i & \alpha(e(at_i)) = \text{true} \\ \neg at_i & \alpha(e(at_i)) = \text{false} \end{cases}$$

and let the set of literals be

$$Th(\alpha) = \{ Th(at_i, \alpha) \mid at_i \in \varphi \}$$

then $\hat{Th}(\alpha)$ is the conjunction of literals in $Th(\alpha)$.

Let DEDUCTION be the procedure DP_T with the possible generation of a blocking clause , $t = \neg \hat{Th}(\alpha)$.

Algorithm 3.3.1: LAZY-BASIC

Input: A formula φ
Output: "Satisfiable" if φ is satisfiable, and "Unsatisfiable" otherwise

```
1. function LAZY-BASIC(φ)
2.     B := e(φ);
3.     while (TRUE) do
4.         ⟨α, res⟩ := SAT-SOLVER(B);
5.         if res ="Unsatisfiable" then return "Unsatisfiable";
6.         else
7.             ⟨t, res⟩ := DEDUCTION(T̂h(α));
8.             if res ="Satisfiable" then return "Satisfiable";
9.             B := B ∧ e(t);
```

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause

The first requirement 1. ensures soundness. The second and third requirements 2. e 3. are sufficient to guaranteeing termination.

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause

The first requirement 1. ensures soundness. The second and third requirements 2. e 3. are sufficient to guaranteeing termination.

Two can be weakened:

- It is enough that t implies φ
- In t can occur other atomic formulas

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause

The first requirement 1. ensures soundness. The second and third requirements 2. e 3. are sufficient to guaranteeing termination.

Two can be weakened:

- It is enough that t implies φ
- In t can occur other atomic formulas

Beside considering an incremental SAT (that keeps the $\mathcal{B}$ from previous calls), it is more efficient to integrate the procedure DEDUCTION in the CDCL algorithm.

Algorithm 3.3.2: LAZY-CDCL

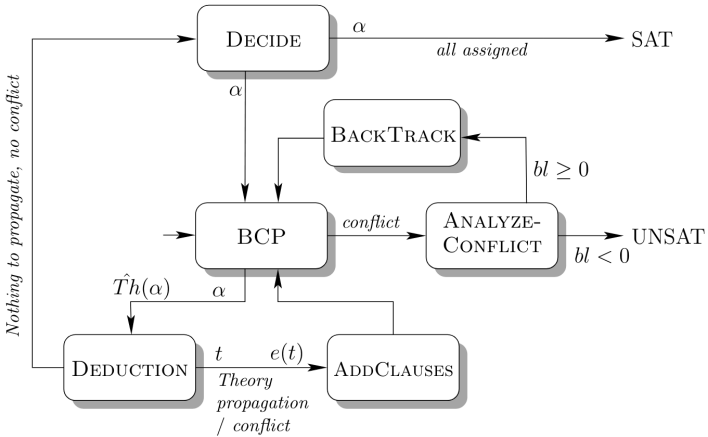
Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsatisfiable" otherwise

```
1.  function LAZY-CDCL
2.      ADDCLAUSES(cnf(e(φ)));
3.      while (TRUE) do
4.          while (BCP() = "conflict") do
5.              backtrack-level := ANALYZE-CONFLICT();
6.              if backtrack-level < 0 then return "Unsatisfiable";
7.              else BackTrack(backtrack-level);
8.          if ¬DECIDE() then                          ▷ Full assignment
9.              ⟨t, res⟩:=DEDUCTION(Ĥ(α));             ▷ α is the assignment
10.             if res="Satisfiable" then return "Satisfiable";
11.             ADDCLAUSES(e(t));
```

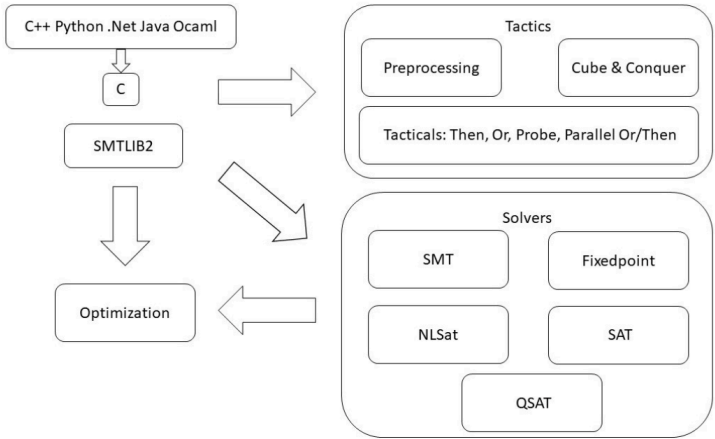
This algorithm uses a procedure ADDCLAUSES, which adds new clauses to the current set of clauses at run time.

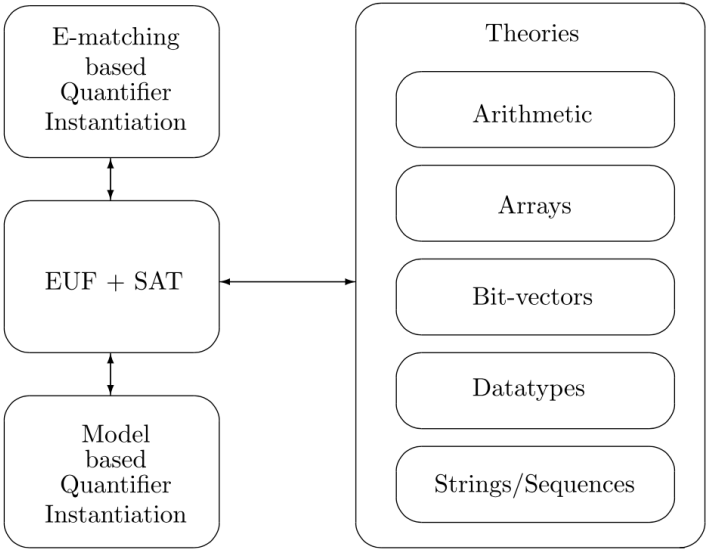
Suppose that φ has an integer variable x_1 and the literals $x_1 < 0$ and $x_1 > 10$. If $e(x_1 > 10) \mapsto$ true and $e(x_1 < 0) \mapsto$ true there will be a contradiction but that is only detected after being obtained a full assignment. However that can be improved, if the call to DEDUCTION is made earlier. That allows to

- Contradictory partial assignments are ruled early
- Implications of literals that are still unassigned can be communicated back to the SAT solver. We call this technique theory propagation.
- For example, if $e(x_1 > 10) \leftarrow$ true we can infer that $e(x_1 < 0) \leftarrow$ false and thus avoid the conflict altogether.


DECIDE
α
all assigned
SAT
α
BACKTRACK
bl ≥ 0
BCP
conflict
ANALYZE-CONFLICT
UNSAT
bl < 0
T̂h(α)
α
DEDUCTION
t
e(t)
Theory propagation / conflict
ADDCLAUSES
Nothing to propagate, no conflict

- Z3 `https://github.com/Z3Prover/z3`
- Z3 `https://z3prover.github.io/papers/programmingz3.html`
- `https://z3prover.github.io/papers/z3internals.html`
- Python : pip install z3-solver
- Tutorial:
  `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`


C++ Python .Net Java Ocaml
C
SMTLIB2
Optimization
Tactics
Preprocessing
Cube & Conquer
Tacticals: Then, Or, Probe, Parallel Or/Then
Solvers
SMT
Fixedpoint
NLSat
SAT
QSAT


E-matching based Quantifier Instantiation
EUF + SAT
Model based Quantifier Instantiation
Theories
Arithmetic
Arrays
Bit-vectors
Datatypes
Strings/Sequences

```
x = Real('x')
y = Real('y')
z = Real('z')
s = Solver()
s.add(3*x + 2*y - z == 1)
s.add(2*x - 2*y + 4*z == -2)
s.add(-x + 0.5*y - z == 0)
print(s.check())
print(s.model())
```

- Logical variables are created indicating their `Sort`: `Real`, `Bool`, `Int`, or any new declared type:

```
S = DeclareSort('S')
f = Function('f', S, S)
x = Const('x', S)
y = Const('y', S)
z  = Const('z', S)
s = Solver()
s.add(Or(x!=y,Or(f(x)==f(y),f(x)!=f(z))))
print(s.check())
print(s.model())
solve(Or(x!=y,Or(f(x)==f(y),f(x)!=f(z)))
```

- `solve()` creates a `Solver`, adds a formula and checks if it is satisfiable returning a solution (`model`).
- `Const` and `Function` define zero or more variables, respectively

- a standard language for SMT is the SMT-LIB (similar to LISP), but we can use the Python interface

```
x, y = Ints('x y')
s = Solver()
s.add((x % 4) + 3 * (y / 2) > x - y)
print(s.sexpr())
```

- outputs

```
(declare-fun y () Int)
(declare-fun x () Int)
(assert (> (+ (mod x 4) (* 3 (div y 2))) (- x y)))
```

- Quantifiers: `ForAll`, `Exists`

```
solve([y == x + 1, ForAll([y], Implies(y <= 0, x < y))])
```

The first occurence of `y` is free, the second is bounded.

```
(set-logic QF UFLIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (distinct x y z))
(assert (> (+ x y) (* 2 z)))
(assert (>= x 0))
(assert (>= y 0))
(assert (>= z 0))
(check-sat)
(get-model)
(get-value (x y z))
```

Usando % z3 exemplo1.smt2

```
sat
(
  (define-fun x () Int
    3)
  (define-fun z () Int
    1)
  (define-fun y () Int
    0)
)
((x 3)
 (y 0)
 (z 1))
```

pyz3: s.from_file("exemplo1.smt2")

- help(class) or help(function)
- *describe_tactics*.
-

- To define arrays one use the (*sort*) `Array`

```
A = Array('A', IntSort(), IntSort())
x, y = Consts('x y',IntSort())
solve(A[x] == x, Store(A, x, y) == A)
```

- $A[x]$ is defined by `Select(A,x)` (or `A[x]`)
- `Store(A,x,v)`, corresponds to $A[x \leftarrow v]$.
- `K(Sort,v)` is an array of `Sort` where all indexes have the value v (constant array, it is used to show a solution).
- For the verification condition above

```
solve (Implies(ForAll([x],(Implies(x< y, A[x]==0))),
 ForAll([x],(Implies(x<= y, Store(A, y, 0)[x]==0)))))
```

Arrays can be represented by λ-terms : if $f : A \times B \to C$ then `Lambda [x,y]. f(x, y)` has type `Array(A,B,C)`.

```
a[i]              # select array 'a' at index 'i'
                  # Select(a, i)
Store(a, i, v)    # update array 'a' with 'v' at index 'i'
                  # = Lambda(j, If(i == j, v, a[j]))
K(D, v)           # constant Array(D, R), where R is sort of 'v'.
                  # = Lambda(j, v)
Map(f, a)         # map function 'f' on values of 'a'
                  # = Lambda(j, f(a[j]))
Ext(a, b)         # Extensionality
                  # Implies(a[Ext(a, b)] == b[Ext(a, b)], a == b)
```

- Many of the theories considered have undecidable fragments when quantifiers are considered.
- Even if those fragments are decidable complexity of the decision procedures is high.
- For quantified boolean formulas (QBF) quantifier elimination is decidable (PSPACE-complete)
- If there are only existential quantifiers (it is a formula in prenex normal form) one can use Skolemization to obtain a equisatisfiable formula
- Some kinds of alternation of quantifiers can also yield decidable fragments of some theories.

- Formulae in prenex normal form
- and in Skolem normal form. Application of Skolemization
- For instance, $\forall y_1 \forall y_2 \exists x.(f(y_1, y_2) \wedge f(x, y_2) \wedge x < 0)$ after Skolemization becomes:

$$\forall y_1 \forall y_2 (f(y_1, y_2) \wedge f(f_x(y_1, y_2), y_2) \wedge f_x(y_1, y_2) < 0)$$

- The general problem is to have a ground formula G which validity must be proven with respect to axioms.
- for instance, prove that

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

is implied by

$$\forall x \forall y . f(x, y) = f(y, x)$$

- Considering the satisfiability problem we need to show that the following formula is unsatisfiable:

$$\forall x \forall y . f(x, y) = f(y, x) \wedge f(h(a), b) \neq f(b, h(a)).$$

- It is easy to see that if x is instantiated to $h(a)$ and y to b we get a contradiction.

- Let $\forall \overline{x}.\psi \wedge G$ be the formula we want to proof unsatisfiable, with G ground.
- One can instantiate $\overline{x}$ with all ground terms of G of the same type
- But that is in general exponential.
- The solver SIMPLIFY implemented an heuristic called E-graph algorithm that is now widely used (and improved):
- for each $\forall \overline{x}.\psi$, identify those subterms in ψ that contain references to all the variables in $\overline{x}$. These are the **triggers**.
- In the example above both $f(x, y)$ and $f(y, x)$ are triggers.
- Try to match each trigger tr (pattern) to an existing ground term gr in G and take the correspondent substitution. In the example, matching $f(x, y)$ to $f(h(a), b)$ yields $\overline{s} = \{x \mapsto h(a), y \mapsto b\}$.
- Assign $G := G \wedge \psi[\overline{x} \leftarrow \overline{s}]$ and check the satisfiability of G.

Example

Consider

$$b = c \implies f(h(a), g(c)) = f(g(b), h(a))$$

where f is commutative, i.e.,

$$\forall x \forall y . f(x, y) = f(y, x)$$

Consider the trigger $f(x, y)$ which can match both $f(h(a), g(c))$, with substitution $\{x \mapsto h(a), y \mapsto g(c)\}$ and $f(g(b), h(a))$ with substitution $\{x \mapsto g(b), y \mapsto h(a)\}$. Then one needs to check the satisfiability of

$$b = c \wedge f(h(a), g(c)) \neq f(g(b), h(a)) \wedge$$
$$f(h(a), g(c)) = f(g(c), h(a)) \wedge f(g(b), h(a)) = f(h(a), g(b))$$

which is unsatisfiable.

- Frequently, however, the predicates necessary for proving unsatisability are not based on terms in the existing formula.
- *Simplify* has a more flexible matching algorithm,
- which exploits its current knowledge on equivalences among various terms, which is called E-matching.
- Based on the syntactic similarity of a trigger (the pattern) tr and a ground term gr
- but also, it can consider any ground term that is known to be equivalent to gr.
- Uses union-find for equivalence classes

Algorithm 9.5.1: E-MATCHING

Input: Trigger tr, term gr, current substitution set sub
Output: Substitution set sub such that for each $\alpha \in sub$, $E \models \alpha(tr) = gr$.

1. **function** MATCH(tr, gr, sub)
2. **if** tr is a variable x **then**
3. **return**
$\{\alpha \cup \{x \mapsto gr\} \mid \alpha \in sub, x \notin dom(\alpha)\} \cup$
$\{\alpha \mid \alpha \in sub, find(\alpha(x)) = find(gr)\}$
4. **if** tr is a constant c **then**
5. **if** $c \in class(gr)$ **then return** sub
6. **else return** $\emptyset$
7. **if** tr is of the form $f(p_1, \ldots, p_n)$ **then return**

$$\bigcup_{f(gr_1,\ldots,gr_n) \in class(gr)} \begin{array}{l} \text{MATCH}(p_n, gr_n, \\ \quad \text{MATCH}(p_{n-1}, gr_{n-1}, \\ \qquad \ddots \\ \qquad\quad \text{MATCH}(p_1, gr_1, sub) \ldots)) \end{array}$$

The algorithm uses union-find to represent the classes of equivalence of equality of terms.

- $dom(\alpha)$ is the domain of the substitution
- $find(gr)$ returns the representative element of the class of gr. If two terms gr_1, gr_2 are such that $find(gr_1) = find(gr_2)$ then it means that they are equivalent.
- $class(gr)$ returns the equivalence class of gr.
- The algorithm uses functional congruence with a member of the equivalence class.
- The output of this algorithm is a set of substitutions, each of which brings us from tr to gr, possibly by using congruence closure.
- If E denotes the equalities, then for each possible substitution $\alpha \in sub$, it holds that $E \models \alpha(tr) = gr$, where $\alpha(tr)$ denotes the substitution applied to the trigger tr.
- For example, for $tr = f(x)$ and $gr = f(a)$, if $E = \{a = b\}$, the value of sub at the end of the algorithm will be $\{x \mapsto a, x \mapsto b\}$.
- If the match fails the empty substitution is returned.

Example

Let $(\forall x.f(x) = x) \wedge (\forall y_1.\forall y_2.g(g(y_1, y_2), y_2) = y_2) \wedge g(f(g(a, b)), b) \neq b$.
The triggers are $f(x)$ and $g(g(y_1, y_2), y_2)$. For the first we consider

$$\text{MATCH}(f(x), f(g(a, b)), \emptyset)$$

Line 7 is invoked and we consider $g(a, b)$:

$$\text{MATCH}(x, g(a, b), \emptyset) = \{x \mapsto g(a, b)\}$$

and $f(g(a, b)) = g(a, b)$ is added to E. For the second trigger $g(g(y_1, y_2), y_2)$, the candidate ground terms for matching are $g(a, b)$ and $g(f(g(a, b)), b))$. In the first case the matching fails

$$\text{MATCH}(y_2, b, \text{MATCH}(g(y_1, y_2), a, \emptyset)) == \mathit{fail}$$

as $\mathit{class}(a)$ has no term with functional symbol g.

In the second case we have

$$
\begin{aligned}
&= \text{MATCH}(y_2, b, \text{MATCH}(g(y_1, y_2), f(g(a, b)), \emptyset)) \\
&= \text{MATCH}(y_2, b, \text{MATCH}(g(y_1, y_2), g(a, b), \emptyset)) \\
&= \text{MATCH}(y_2, b, \text{MATCH}(y_2, b, \text{MATCH}(y_1, a, \emptyset))) \\
&= \text{MATCH}(y_2, b, \text{MATCH}(y_2, b, \{y_1 \mapsto a\})) \\
&= \text{MATCH}(y_2, b, \{y_1 \mapsto a, y_2 \mapsto b\}) \\
&= \{y_1 \mapsto a, y_2 \mapsto b\}
\end{aligned}
$$

Note the switch between $f(g(a, b))$ and $g(a, b)$: it happens, because these two terms are in the same equivalence class according to the E-graph.

As E-matching works only with functional congruence. It cannot deal with interpreted functions (as arithmetic ones).


Nikolai Bjorner and Leonardo de Moura.
Z3 Theorem Prover.
Rise, Microsft, 2015.


Armin Biere, Marjin Heulen, Hans van Maaren, and Tobis Walsh.
Handbook of Satisfiability.
IOS Press, second edition, 2021.


Aaron R. Bradley and Zohar Manna.
The Calculus of Computation: Decision Procedures with Applications to Verification.
Springer Verlag, 2007.


Daniel Kroening and Ofer Strichman.
Decision Procedures:An Algorithmic Point of View.
Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.