Satisfiability Modulo Theories Solvers

- SMT solvers are used as core engines in many tools in
  - program analysis and verification
  - software engineering
  - hardware verification, …
- Combine propositional satisfiability search techniques with solvers for specific first-order theories
  - Linear arithmetic
  - Bit vectors
  - Uninterpreted functions
  - Arrays, …

First-Order Logic

First-Order Logic – Syntax Overview

- Functions , Variables, Predicates
  - *a, b f, g,*   *x, y, z,*   P, Q, =
- Terms
  - a, f(a), g(x, y)
- Atomic formulas, Literals
  - P(x,f(a)), $\neg$Q(y,z)
- Quantifier free formulas
  - P(f(a), b) $\wedge$ c = g(d)
- Formulas, sentences
  - $\forall$x . $\forall$y . [P(x, f(x)) $\vee$ g(y,x) = h(y)]

Signatures

- A *signature* Σ consists of
  - a set of *function symbols*:

    $\Sigma_F = \{ f, g, \ldots \}$
  - a set of *predicate symbols*:

    $\Sigma_P = \{ P, Q, =, \text{true}, \text{false}, \ldots \}$
  - and an *arity* function:

    arity: $(\Sigma_F \cup \Sigma_P) \rightarrow N$
- Function symbols with arity 0 are called *constants*
- A countable set X of *variables*
  - disjoint from Σ

Terms

- Given a signature Σ and a set of variables X
- The set of *terms* $T(\Sigma, X)$ is the smallest set formed by the grammar:

$$
\begin{array}{lll}
t \in T ::= x & & x \in X \\
\quad\quad\;\; | & f(t_1, \ldots, t_n) & f \in \Sigma_F \; t_1, \ldots, t_n \in T
\end{array}
$$

- The terms $T(\Sigma,\emptyset)$ are called *ground terms*.

Atomic Formulas

- *Atomic formulas* are built from terms and predicate symbols:

  $a ::= P(t_1, \ldots, t_n)$ $\quad\quad P \in \Sigma_P \; t_1, \ldots, t_n \in T(\Sigma, X)$

  An atom is *ground* if $t_1, \ldots, t_n \in T(\Sigma_F, \emptyset)$

- Literals are (negated) atoms:

  $l ::= a \mid \neg a$

Quantifier-Free Formulas

- The set QFF(Σ,X) of *quantifier-free formulas* is the smallest set such that:

$\varphi \in \mathrm{QFF}\,(\Sigma, X)$	$::= a$	*atoms*
	$\mid \neg\varphi$	*negations*
	$\mid \varphi \leftrightarrow \varphi'$	*bi-implications*
	$\mid \varphi \wedge \varphi'$	*conjunction*
	$\mid \varphi \vee \varphi'$	*disjunction*
	$\mid \varphi \rightarrow \varphi'$	*implication*

Formulas

- The set of *first-order formulas* are obtained by adding the formation rules:

  φ ::= …

  | $\forall$ x . φ *universal quant.*

  | $\exists$ x . φ *existential quant.*

- *Free occurrences* of variables in a formula are those not bound by a quantifier.
- *A sentence* is a first-order formula with no free variables.

Dreadbury Mansion Mystery

- Someone who lived in Dreadbury Mansion killed Aunt Agatha.
- Agatha, the Butler and Charles were the only people who lived in Dreadbury Mansion.
- A killer always hates his victim, and is never richer than his victim.
- Charles hates no one that aunt Agatha hates.
- Agatha hates everyone except the butler.
- The butler hates everyone not richer than Aunt Agatha.
- The butler also hates everyone Agatha hates.
- No one hates everyone.
- Agatha is not the butler.

Who killed Aunt Agatha?

Semantics: Structures

- A *first-order structure* M consists of:
  - Domain U; nonempty set of elements.
  - Interpretation, $f^M : U^n \rightarrow U$ for each $f \in \Sigma_F$ with arity(f) = n
  - Interpretation $P^M \subseteq U^n$ for each $P \in \Sigma_P$ with arity(P) = n
  - Assignment $x^M \in U$ for every variable $x \in X$

- *A formula* φ is *true* in a structure M if it evaluates to *true* under the given interpretations over the domain U.

Semantics: Evaluation of Terms and Atoms

- A term t in a structure M evaluates to
  - x^M if $t = x$ for some variable $x \in X$
  - $f^M(u_1, \ldots, u_n)$ if $t = f(t_1, \ldots, t_n)$ and each t_i evaluates to u_i in M

- An $P(t_1, \ldots, t_n)$ atom in a structure M evaluates to $b \in \{true, false\}$, where
  - b iff $(u_1, \ldots, u_n) \in P^M$
  - and each t_i evaluates to u_i in M

Semantics: Evaluation of Formulas

- The *satisfaction relation* is defined recursively as follows:
  - $M \models a$ iff a evaluates to *true* in M
  - $M \models \neg\varphi$ iff $M \not\models \varphi$ (M does not satisfy φ)
  - $M \models \varphi \leftrightarrow \varphi'$ iff $M \models \varphi$ is equivalent to $M \models \varphi'$
  - $M \models \varphi \wedge \varphi'$ iff $M \models \varphi$ and $M \models \varphi'$
  - $M \models \varphi \vee \varphi'$ iff $M \models \varphi$ or $M \models \varphi'$
  - $M \models \varphi \rightarrow \varphi'$ iff $M \models \varphi$ implies $M \models \varphi'$
  - $M \models \forall x.\varphi$ iff for all $u \in U$, $M[x \mapsto u] \models \varphi$
  - $M \models \exists x.\varphi$ iff exists $u \in U$, $M[x \mapsto u] \models \varphi$

Notation (as for Propositional Logic)

F, G: first-order formulas over Σ
M: first-order structure over Σ

- $M \models F$ *F* is true in *M* (*M* is a *model* of *F*)
- $\models F$ *F* is valid
- $F \models G$ $F \rightarrow G$ is valid (F *entails* G)
- $F \models \bot$ *F* is unsatisfiable

Semantics: Exercise

- Drinker's paradox:
  - *There is someone in the pub such that, if he is drinking, everyone in the pub is drinking.*
  - $\exists x. (D(x) \rightarrow \forall y. D(y))$

- Is this formula
  - valid?
  - unsatisfiable?
  - satisfiable but not valid?

Theories

- Let $DC(\Gamma)$ be the deductive closure of a set of sentences Γ, i.e. $DC(\Gamma)$ is the smallest set such that
  - $\Gamma \subseteq DC(\Gamma)$
  - $\{\varphi' \mid \varphi \in DC(\Gamma), \varphi \vDash \varphi'\} \subseteq DC(\Gamma)$

- *A* (first-order) *theory* *T* (over signature Σ) is a set of deductively closed sentences (over Σ), i.e. $DC(T) = T$

- If $T = DC(\Gamma)$ for some set of sentences Γ, then the elements of Γ are called *axioms* of *T*

- A theory *T* is *constistent* if *false* $\notin T$

- We can also view a theory *T* as the class of all models of *T*

T-Satisfiability and *T*-Validity

- *M* is a *model for the theory* *T* if all sentences of *T* are *true* in *M.*

- A formula $\varphi(\boldsymbol{x})$ is *T-satisfiable* in a theory *T* if
  - there is a model *M* of *T* in which $\varphi(\boldsymbol{x})$ evaluates to *true*
  - Notation: $M \vDash_T \varphi(\boldsymbol{x})$

- A formula $\varphi(\boldsymbol{x})$ is *T-valid* in a theory *T* if
  - $\varphi(x)$ evaluates to *true* in every model *M* of *T.*
  - Notation: $\vDash_{\mathrm{T}} \varphi(\boldsymbol{x})$

Theory of Equality T_E

- also known as *theory of uninterpreted functions* and *theory of free functions*

- Signature: $\Sigma_E = \{ =, a, b, c, \ldots, f, g, h, \ldots, P, Q, R, \ldots. \}$
  - = is a binary predicate, interpreted by axioms
  - all constant, function, and predicate symbols.

- Axioms:

1. $\forall x \,.\, x = x$ (reflexivity)
2. $\forall x, y.\ x = y \rightarrow y = x$ (symmetry)
3. $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$ (transitivity)

Theory of Equality T_E

- Axioms (continued):

4. for each positive integer *n* and *n*-ary function symbol *f*,

$$\forall x_1,\ldots, x_n, y_1,\ldots, y_n .\ \bigwedge_i x_i = y_i \rightarrow f(x_1,\ldots, x_n) = f(y_1,\ldots, y_n)$$

(congruence)

5. for each positive integer *n* and *n*-ary predicate symbol *P*

$$\forall x_1,\ldots, x_n, y_1,\ldots, y_n .\ \bigwedge_i x_i = y_i \rightarrow (P(x_1,\ldots, x_n) \leftrightarrow P(y_1,\ldots, y_n))$$

(equivalence)

Theory Example: Peano Arithmetic (Natural Numbers)

- Signature: Σ_{PA} = { 0, 1, +, ∗, = }
- Axioms of T_{PA} : axioms for theory of equality, T_E , plus:

1. ∀x. ¬ (x + 1 = 0) (zero)
2. ∀x, y. x + 1 = y + 1 → x = y (successor)
3. *F*[0] ∧ (∀*x*.*F*[*x*] → *F*[*x* + 1]) → ∀*x*.*F*[*x*] (induction)
4. ∀x. x + 0 = x (plus zero)
5. ∀x, y. x + (y + 1) = (x + y) + 1 (plus successor)
6. ∀x. x ∗ 0 = 0 (times zero)
7. ∀x, y. x ∗ (y + 1) = x ∗ y + x (times successor)

Line 3 is an axiom schema for all formulas *F*[*x*].

Theory Fragments

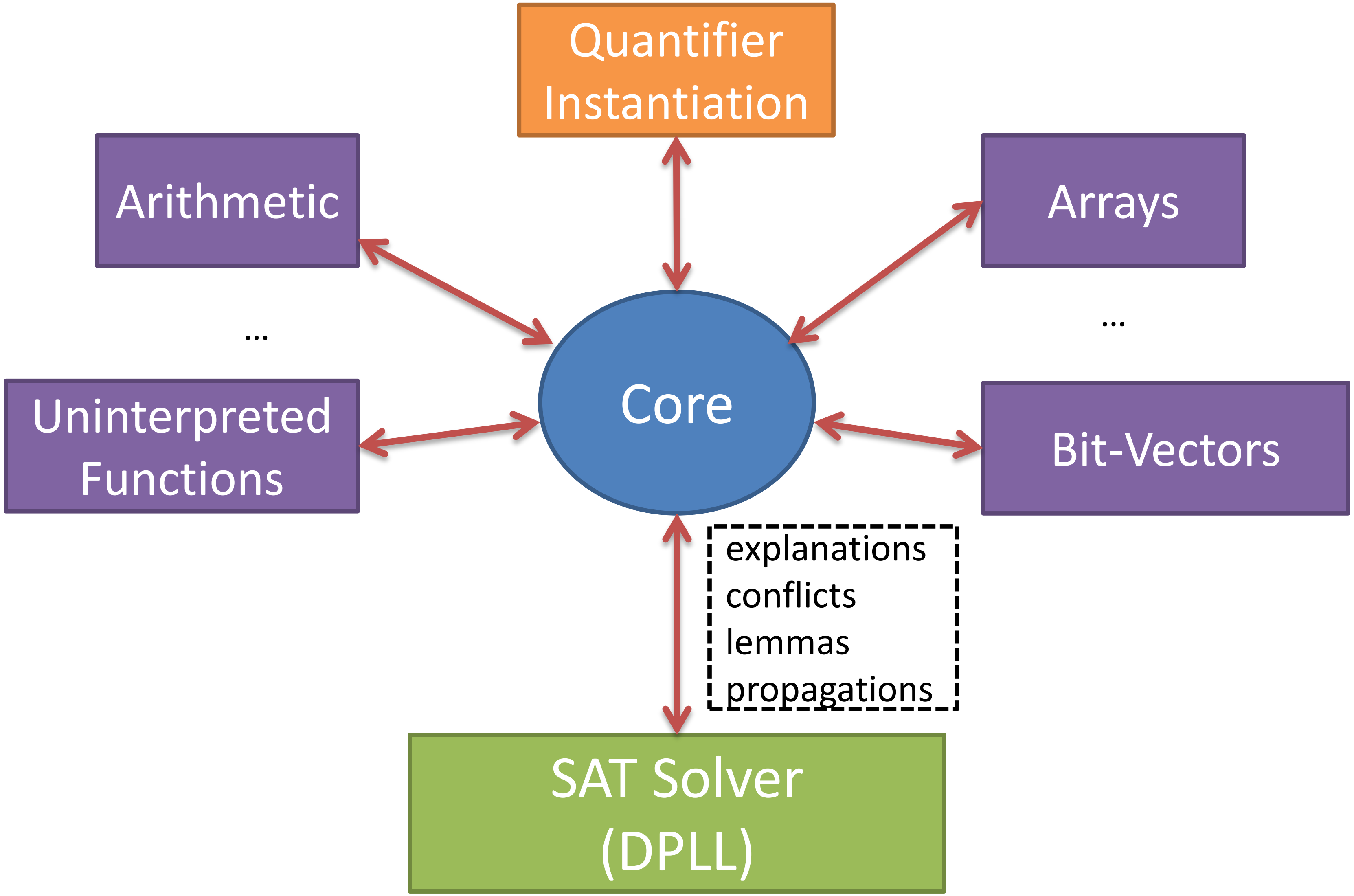
- A *fragment* of a theory T is a syntactically restricted subset of the formulas of the theory
- Example:
  - The *quantifier-free fragment* of theory T is the set of formulas without quantifiers that are valid in T
- Often there are decidable fragments for undecidable theories
- Theory T is *decidable* if T-validity is decidable for every formula F of T
  - There is an algorithm that always terminates with "yes" if F is T-valid, and "no" if $\neg F$ is T-satisfiable

Theory Fragments: Examples

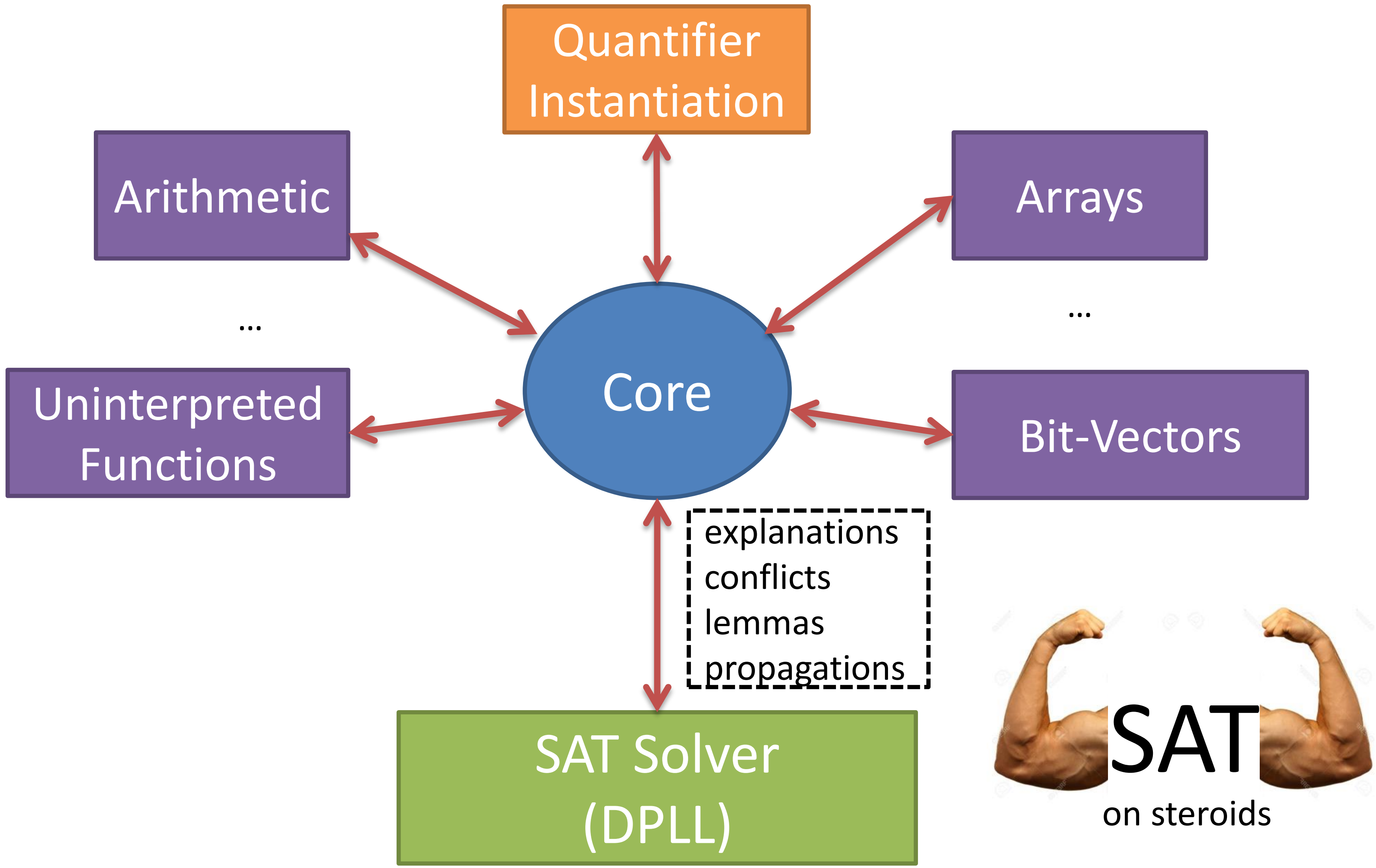
- The theory of equality is undecidable
  - its quantifier-free fragment is decidable
  - its fragment consisting of formulas of the form $\exists \boldsymbol{y}.\ \forall \boldsymbol{x}.\ F(\boldsymbol{x},\boldsymbol{y})$ where F is quantifier-free and the variables $\boldsymbol{x}$ do not appear below function symbols is decidable (Bernays-Schoenfinkel-Ramsey fragment)
- The theory of integer arithmetic is undecidable
  - the theory of linear integer arithmetic is decidable (Presburger arithmetic)
- The theory of arithmetic over reals is decidable

SMT Solvers

SMT Solver Architecture



SMT Solver Architecture



SMT-LIB Syntax

```
(set-logic QFUFLIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (>= (* 2 x) (+ y z)))
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
```

- choose logic/theories
- declare signature
- assert formula
- check satisfiability

SMT-LIB Syntax

```
...
(check-sat)
(get-model)
(push)
(assert (= x y))
(check-sat)
(pop)
(exit)
```

clauses can be added and removed interactively

Lazy Approach to SMT

Lazy Approach to SMT

$$g(a) = c \;\wedge\; (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$$

Lazy Approach to SMT

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

$$1 \wedge (\neg 2 \vee 3) \wedge \neg 4$$

Propositional abstraction

Lazy Approach to SMT

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

- SAT solver returns model [1, ¬2, ¬4]

Lazy Approach to SMT

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

- SAT solver returns model [1, ¬2, ¬4]
- Theory solver detects [1, ¬2] T-unsat

Lazy Approach to SMT

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

- SAT solver returns model [1, ¬2, ¬4]
- Theory solver detects [1, ¬2] T-unsat
- Send [1, ¬2 ∨ 3, ¬4, ¬1 ∨ 2] to SAT solver

theory lemma

Lazy Approach to SMT

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

- SAT solver returns model [1, $\neg$2, $\neg$4]
- Theory solver detects [1, $\neg$2] T-unsat
- Send [1, $\neg$2 $\vee$ 3, $\neg$4, $\neg$1 $\vee$ 2] to SAT solver
- SAT solver returns model [1, 2, 3, $\neg$4]

Lazy Approach to SMT

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

- SAT solver returns model [1, ¬2, ¬4]
- Theory solver detects [1, ¬2] T-unsat
- Send [1, ¬2 ∨ 3, ¬4, ¬1 ∨ 2] to SAT solver
- SAT solver returns model [1, 2, 3, ¬4]
- Theory solver detects [1, 3, ¬4] T-unsat

Lazy Approach to SMT

$$g(a) = c \;\wedge\; (f(g(a)) \neq f(c) \vee g(a) = d) \;\wedge\; c \neq d$$

$$1 \;\wedge\; (\neg 2 \;\vee\; 3) \;\wedge\; \neg 4$$

- SAT solver returns model [1, ¬2, ¬4]
- Theory solver detects [1, ¬2] T-unsat
- Send [1, ¬2 ∨ 3, ¬4, ¬1 ∨ 2] to SAT solver
- SAT solver returns model [1, 2, 3, ¬4]
- Theory solver detects [1, 3, ¬4] T-unsat
- Send [1, ¬2∨3, ¬4, ¬1∨2, ¬1∨¬3∨4] to SAT solver

Lazy Approach to SMT

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

- SAT solver returns model [1, $\neg$2, $\neg$4]
- Theory solver detects [1, $\neg$2] T-unsat
- Send [1, $\neg 2 \vee 3$, $\neg$4, $\neg 1 \vee 2$] to SAT solver
- SAT solver returns model [1, 2, 3, $\neg$4]
- Theory solver detects [1, 3, $\neg$4] T-unsat
- Send [1, $\neg 2 \vee 3$, $\neg$4, $\neg 1 \vee 2$, $\neg 1 \vee \neg 3 \vee 4$] to SAT solver
- SAT solver detects unsat

Lazy Approach to SMT

- SAT solver handles all propositional reasoning
- Theory solvers only need to reason about conjunctions of literals
  - How to decide T-satisfiability of individual theories?
  - How to compose individual theory solvers to decide theory combinations?
  - How to deal with quantifiers?