CS257: Introduction to Automated Reasoning

Quantifier Instantiation

Stanford University

CENTAUR

SMT solvers

- Traditionally:
  - Efficient decision procedures for quantifier-free constraints over theories:
  - Arithmetic
  - Uninterpreted functions (UF)
  - Bitvectors
  - Arrays
  - Datatypes
  - More recently: strings, floating points, sets, relations, ...
- In the past decade or so:
  - Efficient (heuristic) techniques for quantified formulas as well
  - Focus of this lecture.

Applications of $\forall$ in SMT

Quantifiers are used for:

- Automated theorem proving:
  - Background axioms: $\forall x, y.(x + y = y + x)$
- Software verification:
  - Unfolding: $\forall x.(foo(x) = bar(x + 1))$
  - Code contracts: $\forall x.(pre(x) \rightarrow post(f(x)))$
  - Frame axioms: $\forall x.(x > 0 \rightarrow f(x) = f(x + 1))$
- Function synthesis:
  - Synthesis conjectures: $\forall i : input.\exists o : output.R(o, i)$
- Planning:
  - Specifications: $\exists p : plan.\forall t : time.R(p, t)$

Today

- Herbrand Theorem
- Quantifier Instantiation (DP Ch. 9.5)
    - Trigger-based instantiation strategies
    - Other instantiation strategies:
        - conflict-based instantiation
        - model-based instantiation

Some of the slides are contributed by Andrew Reynolds.

Review: Clausal Form

We say a first-order logic formula is in **Clausal Form** if,

1. it is in PCNF;
2. it is closed (i.e., does not contain free variables); and
3. it only contains universal quantifiers.

Example: $\forall y. \forall z. (p(f(y)) \wedge \neg q(y, z))$

Given any first-order logic sentence ϕ, one can transform ϕ into an equi-satisfiable formula ϕ' in clausal form

Example: $\forall x. (p(x) \rightarrow \exists y. q(x, y))$

1. Eliminate implications: $\forall x. (\neg p(x) \vee \exists y. q(x, y))$
2. Skolemize ($y \mapsto f_y(x)$): $\forall x. (\neg p(x) \vee q(x, f_y(x)))$

First-order satisfiability

Skolemization reduces the problem of first-order satisfiability to first-order satisfiability of formulas in clausal form

Herbrand's Theorem will further reduce this (in a weaker sense) to propositional satisfiability

For now, assume we are dealing with formulas in clausal form

Herbrand Interpretation

Given a Σ-formula ϕ, e.g.,

$$\forall x. \, (\neg p(x) \vee q(x, g(x)))$$

there is no easy way to describe the set of possible interpretations (e.g., the definitions of p, q, g can be arbitrary)

We define canonical interpretations called **Herbrand interpretations**, which have the following property:

if ϕ is satisfiable, then there is a Herbrand interpretation that satisfies ϕ

For simplicity, consider a signature $\Sigma := \{\Sigma^S, \Sigma^F\}$ without equality, with one sort S (other than Bool), and assume the arguments of function symbols have sort S:

- For $f \in \Sigma^F$, either $sort(f) = \langle S, \ldots, S \rangle$ or $sort(f) = \langle S, \ldots, S, \mathrm{Bool} \rangle$

Herbrand Interpretation: domain

The first thing that an interpretation needs is the domain of sort S

Given a formula ϕ. Let $\mathcal{A}$ be the set of constant symbols in ϕ, and $\mathcal{F}$ be the set of function symbols that have positive arities and return S

The **Herbrand universe** of ϕ, H_ϕ, is the set of well-sorted terms generated by $\mathcal{F}$ from $\mathcal{A}$

If there are no constant symbols, initialize $\mathcal{A}$ with an arbitrary symbol a of sort S

Example: Consider formula $\phi := \forall x. \forall y. \Delta$, what is the Herbrand universe when:

- $\Delta := \{\{p(a), \neg p(b), q(x)\}, \{\neg p(b), \neg q(y)\}\}$
  - $H_\phi = \{a, b\}$
- $\Delta := \{\{\neg p(x, f(y))\}, \{p(x, g(x))\}\}$
  - $H_\phi = \{a, f(a), g(a), f(f(a)), f(g(a)), g(f(a)), g(g(a)), \ldots\}$
- $\Delta := \{\{\neg p(a, f(x, y))\}, \{p(b, f(x, y))\}\}$
  - $H_\phi = \{a, b, f(a, a), f(a, b), f(b, a), f(b, b), f(a, f(a, a)), \ldots\}$

Herbrand Interpretation: functions

The Herbrand universe, H_ϕ is the domain of S in a Herbrand interpretation

Now that we have a domain, we need to define the function symbols:

- non-predicate functions: Define $a^{\mathcal{I}}$ as $a \in H_\phi$, define $f^{\mathcal{I}}(a)$ as $f(a) \in H_\phi$
- Predicate symbols: can be defined arbitrarily (i.e., arbitrary relations of the appropriate arities over H_ϕ)

Herbrand Bases and ground instances

An alternative way to view predicate symbols is through the lens of a **Herbrand base**

Given a formula α, a **ground instance** of α is the result of replacing every free variable in α with an element of the Herbrand universe H_ϕ

The **Herbrand base** for ϕ, B_ϕ, is the set of ground instances of atomic formulas in ϕ

Example: Consider the third example from the previous slide

$$\begin{aligned}
\phi &:= \{\{\neg p(a, f(x, y))\}, \{p(b, f(x, y))\}\} \\
H_\phi &:= \{a, b, f(a, a), f(a, b), f(b, a), f(b, b), f(a, f(a, a)), \ldots\} \\
B_\phi &:= \{p(a, f(a, a)), p(a, f(a, b)), p(a, f(b, a)), p(a, f(b, b)), \ldots \\
&\quad p(b, f(a, a)), p(b, f(a, b)), p(b, f(b, a)), p(b, f(b, b)) \ldots\}
\end{aligned}$$

A predicate symbol in a Herbrand interpretation can be defined as a subset of B_ϕ, containing those instances of the predicate which evaluate to $\mathtt{T}$

For example, $\{p(b, f(a, a)), p(b, f(a, b)), p(b, f(b, a)), p(b, f(b, b))\}$

Note: we call a formula/term that does not contain variables a **ground formula/term**

Herbrand Bases and ground instances

An alternative way to view predicate symbols is through the lens of a **Herbrand base**

Given a formula α, a **ground instance** of α is the result of replacing every free variable in α with an element of the Herbrand universe H_ϕ

The **Herbrand base** for ϕ, B_ϕ, is the set of ground instances of atomic formulas in ϕ

Exercise: What is the Herbrand base of the following formula:

$$\begin{aligned} \phi &:= \{\{\neg p(x, f(y))\}\} \\ H_\phi &:= \{a, f(a), f(f(a)), \ldots\} \\ B_\phi &:= ? \end{aligned}$$

Submit your answers to

`https://pollev.com/andreww095`

Herbrand Models are Canonical

Theorem: if ϕ (a formula in clausal form) is satisfiable, then there is a Herbrand interpretation $\mathcal{I}$ that satisfies ϕ

Note: $\mathcal{I}$ (first-order) satisfies $\phi := \forall \overline{x}.\Delta$ iff every ground instance of Δ is satisfied by $\mathcal{I}$

Proof sketch: Let J be an interpretation s.t. $J \vDash \phi$, we define a Herbrand interpretation $\mathcal{I}$ based on J and show that $\mathcal{I} \vDash \phi$.

We only need to define $R^{\mathcal{I}}$ for each predicate symbol R in ϕ Let e^J be the evaluation function associated with J. Recall

- For each variable v, $e^J(v) = v^J$.
- If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then $e^J(ft_1, \ldots, t_n) = f^J(e^J(t_1), \ldots, e^J(t_n))$.

We define $R^{\mathcal{I}}$ by the following subset of Herbrand base

$$\{R(t_1, \ldots, t_n) \mid R^J(e^J(t_1), \ldots, e^J(t_n)) = \mathtt{T}\}$$

One can then show that $\mathcal{I} \vDash \phi$.

Details can be found in Chap. 9.3 of "Mathematical Logic for Computer Science" by Ben-Ari

Herbrand's Theorem

We say a quantifier-free sentence is **propositionally satisfiable** if its boolean skeleton is satisfiable

Theorem: A formula $\phi := \forall \overline{x}.\Delta$ is first-order satisfiable iff the set of all ground instances of Δ is (simultaneously) propositionally satisfiable.

Proof: Suppose ϕ is first-order satisfiable. Then there is some Herbrand interpretation $\mathcal{I}$ s.t. $\mathcal{I} \vDash \phi$. For each ground instance gr of an atomic formula in Δ, we associate it with a propositional variable p_{gr}. We give a variable assignment d over the set of all such propositional variables based on $\mathcal{I}$. In particular, $d(p_{gr}) = \mathtt{T}$ iff $e^{\mathcal{I}}(gr) = \mathtt{T}$.

We show that d propositionally satisfies any ground instance Δ_0 of Δ.

By definition of first-order satisfiability, $\mathcal{I}$ satisfies Δ_0, and for each (ground) clause C in Δ_0, there is a (ground) literal ℓ that is satisfied by $\mathcal{I}$. This means the propositional literal corresponding to ℓ must evaluate to $\mathtt{T}$ under d. Thus, d satisfies the boolean skeleton of C, and in turn, of Δ_0.

Herbrand's Theorem

Theorem: A formula $\phi := \forall \overline{x}.\Delta$ is first-order satisfiable iff the set of all ground instances of Δ is (simultaneously) propositionally satisfiable.

Proof (continued): Conversely, suppose d is a variable assignment propositionally satisfying all ground instances of Δ.

We can define a Herbrand interpretation $\mathcal{I}$ using the following subset of the Herbrand base: $\{gr \mid d(p_{gr}) = \mathtt{T}, gr \in B_{\phi}\}$

We claim that $\mathcal{I} \models \phi$. That is, any ground instance Δ_0 is satisfied by $\mathcal{I}$.

This is true because for any (ground) clause C in Δ_0, there must be a literal ℓ whose corresponding propositional literal evaluates to $\mathtt{T}$ under d, which means ℓ is satisfied and in turn C satisfied. $\square$

Herbrand's Theorem

Compactness Theorem of Propositional Logic: a set of propositional logic formula is satisfiable iff every finite subset of it is satisfiable.

The following corollary follows from the Compactness Theorem.

Corollary: A formula $\phi := \forall \overline{x}.\Delta$ is first-order satisfiable iff every finite set of ground instances of Δ is propositionally satisfiable.

Herbrand's Theorem (second form): A formula $\phi := \forall \overline{x}.\Delta$ is first-order unsatisfiable iff some finite set of ground instances of Δ is propositionally unsatisfiable.

Herbrand's Theorem

Herbrand's Theorem (second form): A formula $\phi := \forall \overline{x}.\Delta$ is first-order unsatisfiable iff some finite set of ground instances of Δ is propositionally unsatisfiable.

This leads naturally to a procedure for proving the unsatisfiability of ϕ

We can enumerate larger and larger sets of ground instances of Δ and test them for propositional satisfiability

If we find a set of ground instances that is propositionally unsatisfiable, then ϕ is first-order unsatisfiable

This process of generating ground instances to check for satisfiability is called **quantifier instantiation**

This is (basically) how quantifiers are handled by SMT solvers!

Note: if we guarantee that all finite sets of ground instances are eventually tried, then this gives us a semi-decision procedure for validity of first-order formulas

Quantifier Instantiation in SMT solvers

Quantifiers in formulas are generally handled by SMT solvers through **instantiations**

capitalizing on their capability to handle large ground formulas

Note: we will focus on the case where the background theory is $T_=$, the theory of uninterpreted functions with equality

So far, we focused on the scenario of checking the satisfiability of a single formula in clausal form

Let us switch viewpoints and consider a more typical scenario in SMT: we want to check the satisfiability of a set of ground formulas E in conjunction with a set of quantified formulas Q (in clausal form)

To prove unsatisfiability, try to generate a set of ground formulas E' by instantiating the universally quantified variables in Q in order to reach a contradiction with E

An instantiation can be defined by a **substitution**, a mapping from variables to ground terms

Quantifier Instantiation: Motivating Example

Suppose we want to prove

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

under the assumption that

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

Presenting this as a satisfiability problem, we need to show that the following formula is unsatisfiable:

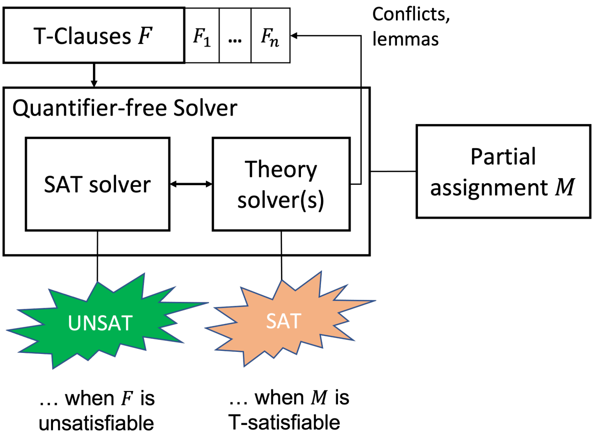
$$\forall x. \forall y. (f(x, y) = f(y, x)) \quad \wedge \quad f(h(a), b) \neq f(b, h(a))$$

What should we instantiate x and y with? $\{x \mapsto h(a), y \mapsto b\}$

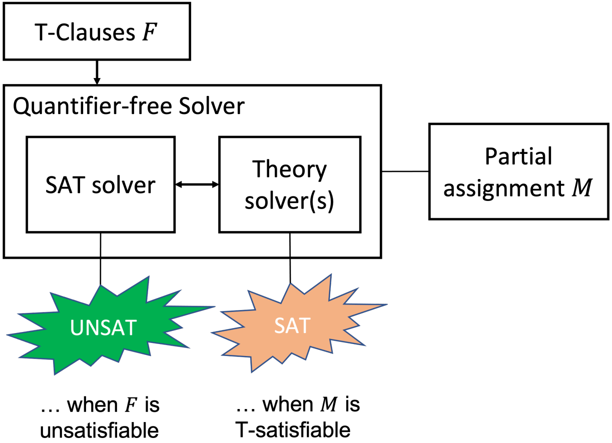
Check $T_=$-satisfiability of

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

DPLL(T)-Based SMT Solvers

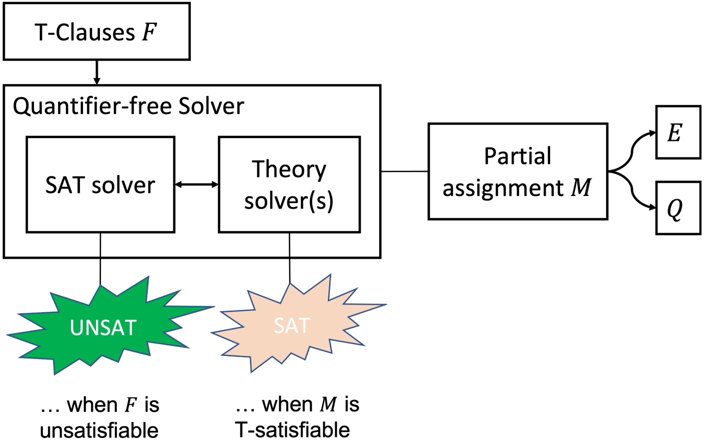


DPLL(T)-Based SMT Solvers + ∀ Instantiation



When M contains quantified formulas...
...cannot use quantifier-free solver for establishing M is sat

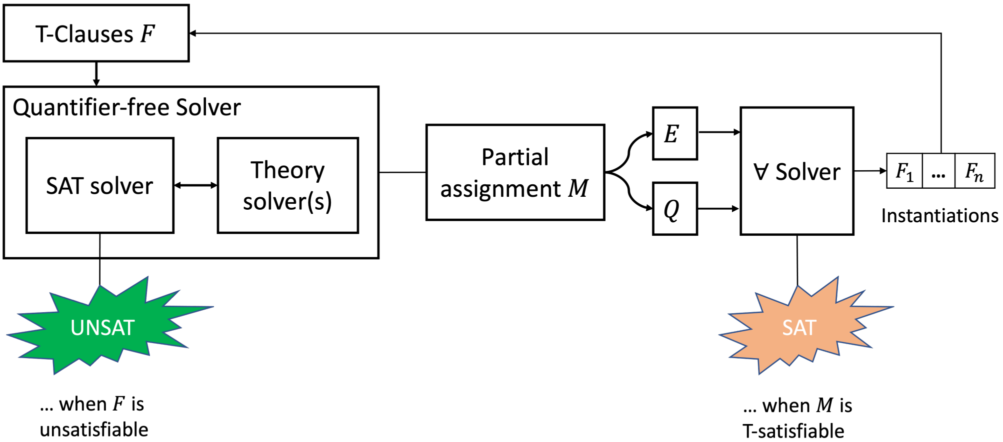
DPLL(T)-Based SMT Solvers + $\forall$ Instantiation



Ground formulas:
e.g., $f(a) = b, P(a) = \top$

Quantified formulas:
e.g., $\forall x.P(x)$

DPLL(T)-Based SMT Solvers + ∀ Instantiation



Quantifier Instantiation: Motivating Example

We wanted to show that the following formula is unsatisfiable:

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

One successful instantiation substitutes x with $h(a)$, and y with b

In principle, to find a successful instantiation, we could enumerate the corresponding Herbrand universe, but it is too large.

It seems to be a good idea to limit ourselves to terms already in E

Quantifier Instantiation: Strategies

Let $\forall \overline{x}.\psi \wedge E$ be the formula that we attempt to prove to be unsatisfiable

A naïve strategy: instantiate $\overline{x}$ with all the terms in E of the same sort

Can lead to an exponential number (in $|\overline{x}|$) of added ground terms

For example:

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

x and y can be instantiated with $a, b, h(a), f(h(a),b), f(b,h(a))$, yielding 25 new predicates

Quantifier Instantiation: Strategies

A better strategy: instantiate $\overline{x}$ to match existing terms in E

- For a quantified formula $\forall \overline{x}.\psi$, select subterms $\{t_1, \ldots, t_n\}$ in ψ that contain references to all variables in $\overline{x}$
  - these terms are called **triggers**
  - In $\forall x.\forall y.(f(x,y) = f(y,x))$, both $f(x,y)$ and $f(y,x)$ can be triggers
- Try to **match** a trigger tr to an existing ground term gr in E
  - Matching $f(x,y)$ to $f(h(a),b)$ yields the substitution $s = \{x \mapsto h(a), y \mapsto b\}$
- Check the satisfiability of $\psi[s] \wedge B$
  - $\psi[s]$ denotes the ground formula resulting from substituting s for $\overline{x}$ in ψ

Example

Suppose we want to prove

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

under the same assumption that

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

Cast in terms of satisfiability, we need to prove the unsatisfiability of

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

Select $f(x, y)$ as the trigger. Can match $f(x, y)$ to $f(h(a), g(c))$ with the substitution $\{x \mapsto h(a), y \mapsto g(c)\}$ or to $f(g(b), h(a))$ with $\{x \mapsto g(b), y \mapsto h(a)\}$. Now we check the $T_=$-satisfiability of

$$\begin{aligned} f(h(a), g(c)) = f(g(c), h(a)) \quad \wedge \\ f(g(b), h(a)) = f(h(a), g(b)) \quad \wedge \\ b = c \quad \wedge \quad f(h(a), g(c)) \neq f(g(b), h(a)) \end{aligned}$$

Example (cont.)

Now we check the $T_=$-satisfiability of

$$\begin{aligned} f(h(a), g(c)) = f(g(c), h(a)) \quad &\wedge \\ f(g(b), h(a)) = f(h(a), g(b)) \quad &\wedge \\ b = c \quad \wedge \quad f(h(a), g(c)) \neq f(g(b), h(a)) & \end{aligned}$$

Unsatisfiable: thus the instantiation is successful

In fact, the first substitution is already enough

How eagerly we should add the terms is a heuristic choice

Quantifier Instantiation: Strategies

Current strategy: instantiate $\overline{x}$ to match existing terms in E

Sometimes, the instantiations necessary for proving unsatisfiability are not based on terms in the existing formulas

Consider the formula

$$\forall x.p(x,b) \quad \wedge \quad b = c \quad \wedge \quad \neg p(a,c)$$

Suppose we select trigger $p(x,b)$, we cannot match it with any ground terms

A successful instantiation would be $p(a,b)$

A more flexible matching strategy (**E-Matching**): find a substitution s for trigger tr, such that $E \models_{=} tr[s] = gr$ for some ground term gr in E

Need knowledge about equalities between terms in E, which can be obtained with the Congruence Closure algorithm

E-Matching: Challenges

- Too many instances
  - Typical real problems: hundreds of $\forall$ in Q, and thousands of terms in E
  - Can add millions of ground instances
  - Need heuristics to select triggers and control eagerness
- Incompleteness
  - $(\forall x.(f(2x - x) < x)) \wedge (f(a) \geq a)$
    Without rewriting $2x - x$ to x, E-Matching cannot find the correct instantiation
  - $(\forall x.f(x) = f(g(x))) \wedge f(g(a)) = a$
    Can get stuck in infinite loops and cannot conclude sat

Beyond E-Matching

Challenges

- Too many instances
- Incompleteness

Many techniques have been proposed to tackle the above two challenges.
We briefly survey two of them:

- Conflict-based instantiation [Reynold'2014]
- Model-based instantiation [Ge'2009]

Conflict-based Instantiation

Search for one instance of one quantified formula in Q that makes E unsatisfiable

- $E = \{\neg P(a), \neg P(b), P(c), \neg R(b)\}$ and $Q = \{\forall x.(P(x) \vee R(x))\}$
- Since $E, P(b) \vee R(b) \models \bot$, returns $x \mapsto b$
- More generally, given $E, \forall \overline{x}.\phi$ returns s s.t. $E \models \neg\phi[s]$ or $\varnothing$ otherwise
- Detecting such conflicts can be computationally expensive (NP-Complete)
- In practice, only look for "shallow" conflicts and avoid exponential behaviors

Reynolds et al. "Finding Conflicting Instances of Quantified Formulas in SMT", FMCAD, 2014

Model-based Instantiation

If E is T-satisfiable, build a candidate interpretation $\mathcal{I}$ where $\mathcal{I} \models E$

check if M also satisfies Q using a quantifier-free satisfiability query

Gives us ability to answer "sat"

- $E = \{\neg P(a), P(b), \neg R(b), \neg R(c), R(a)\}$ and
  $Q = \{\forall x.(P(x) \vee R(x))\}$
- $P^{\mathcal{I}} := \texttt{ite}(x = a, \bot, \texttt{ite}(x = b, \top, \texttt{ite}(x = c, \top, \top)))$
  $R^{\mathcal{I}} := \texttt{ite}(x = a, \top, \texttt{ite}(x = b, \bot, \texttt{ite}(x = c, \bot, \top)))$
- Check satisfiability of $\neg(P^{\mathcal{I}}(x) \vee R^{\mathcal{I}}(x))$
- If unsatisfiable, $\mathcal{I}$ also satisfies Q
- If satisfiable, refine the model with the counter-example found and try again

Ge and de Moura. "Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories", CAV, 2009

Quantifier Instantiation: Summary

In practice, all the aforementioned strategies are used. One possible order is the following:

1. Conflict-based instantiation
   if successful, return UNSAT, otherwise, go to step 2
2. E-matching
   check the resulting ground formulas E and construct candidate model $\mathcal{I}$
3. Model-based instantiation
   check whether $\mathcal{I}$ is a model for both E and Q

Other instantiation strategies exist:

- Counter-example guided:
  Reynolds et al. "Counterexample-Guided Quantifier Instantiation for Synthesis in SMT", CAV 2015
- Enumeration-based:
  Reynolds et al. "Revisiting Enumerative Instantiation", TACAS 2018