Verification

Lecture 24

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

Combining Decision Procedures

Given

Theories T_i over signatures Σ_i
(constants, functions, predicates)
with corresponding decision procedures P_i for T_i-satisfiability.

Goal

Decide satisfiability of a sentence in theory $\bigcup_i T_i$.

Example: How do we show that

$$F : \ 1 \leq x \ \wedge \ x \leq 2 \ \wedge \ f(x) \neq f(1) \ \wedge \ f(x) \neq f(2)$$

is $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable?

Combining Decision Procedures

Σ_1-theory T_1

$\boxed{P_1}$ for T_1-satisfiability

Σ_2-theory T_2

$\boxed{P_2}$ for T_2-satisfiability

?

$\boxed{P}$ for $(T_1 \cup T_2)$-satisfiability

Problem:

Decision procedures are domain specific.

How do we combine them?

Nelson-Oppen Combination Method (N-O Method)

$$\Sigma_1 \cap \Sigma_2 = \{=\}$$

Σ_1-theory T_1
stably infinite

$\boxed{P_1}$ for T_1-satisfiability
of quantifier-free Σ_1-formulae

Σ_2-theory T_2
stably infinite

$\boxed{P_2}$ for T_2-satisfiability
of quantifier-free Σ_2-formulae

$\boxed{P}$ for $(T_1 \cup T_2)$-satisfiability
of quantifier-free $(\Sigma_1 \cup \Sigma_2)$-formulae

Nelson-Oppen: Limitations

Given formula F in theory $T_1 \cup T_2$.

1. F must be quantifier-free.
2. Signatures Σ_i of the combined theory only share $=$, i.e.,

$$\Sigma_1 \cap \Sigma_2 = \{=\}$$

3. Theories must be stably infinite.

Note:

- Algorithm can be extended to combine arbitrary number of theories T_i --- combine two, then combine with another, and so on.
- We restrict F to be conjunctive formula --- otherwise convert to DNF and check each disjunct.

Stably Infinite Theories

A Σ-theory T is stably infinite iff
for every quantifier-free Σ-formula F:
if F is T-satisfiable
then there exists some T-interpretation that satisfies F
and that has a domain of infinite cardinality.

Example: Σ-theory T

$$\Sigma : \{a, b, =\}$$

Axiom

$$\forall x.\, x = a \ \vee \ x = b$$

For every T-interpretation I, $|D_I| \leq 2$ (at most two elements).
Hence, T is not stably infinite.

All the other theories mentioned so far are stably infinite.

Example: Theory of partial orders

Σ-theory $T_{\preceq}$

$$\Sigma_{\preceq} : \{\preceq, =\}$$

where $\preceq$ is a binary predicate.

Axioms

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

We prove $T_{\preceq}$ is stably infinite.

Consider $T_{\preceq}$-satisfiable quantifier-free $\Sigma_{\preceq}$-formula F.
Consider arbitrary satisfying $T_{\preceq}$-interpretation $I : (D_I, \alpha_I)$,
where α_I maps $\preceq$ to $\preceq_I$.

- Let A be any infinite set disjoint from D_I
- Construct new interpretation $J : (D_J, \alpha_J)$
  - $D_J = D_I \cup A$
  - $\alpha_J = \{\preceq \mapsto \preceq_J\}$, where for $a, b \in D_J$,

$$a \preceq_J b \stackrel{\text{def}}{=} \begin{cases} a \preceq_I b & \text{if } a, b \in D_I \\ a = b & \text{otherwise} \end{cases}$$

J is $T_{\preceq}$-interpretation satisfying F with infinite domain.
Hence, $T_{\preceq}$ is stably infinite.

Example: Consider quantifier-free conjunctive $(\Sigma_E \cup \Sigma_{\mathbb{Z}})$-formula

$$F:\ 1 \leq x \ \wedge\ x \leq 2 \ \wedge\ f(x) \neq f(1) \ \wedge\ f(x) \neq f(2)\ .$$

The signatures of T_E and $T_{\mathbb{Z}}$ only share $=$. Also, both theories are stably infinite. Hence, the N-O combination of the decision procedures for T_E and $T_{\mathbb{Z}}$ decides the $(T_E \cup T_{\mathbb{Z}})$-satisfiability of F.

Intuitively, F is $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable.
For the first two literals imply $x = 1 \ \vee\ x = 2$ so that
$f(x) = f(1) \ \vee\ f(x) = f(2)$.
Contradict last two literals.
Hence, F is $(T_E \cup T_{\mathbb{Z}})$-unsatisfiable.

N-O Overview

Phase 1: Variable Abstraction

- Given conjunction Γ in theory $T_1 \cup T_2$.
- Convert to conjunction $\Gamma_1 \cup \Gamma_2$ s.t.
  - Γ_i in theory T_i
  - $\Gamma_1 \cup \Gamma_2$ satisfiable iff Γ satisfiable.

Phase 2: Check

- If there is some set S of equalities and disequalities between the shared variables of Γ_1 and Γ_2
  $\text{shared}(\Gamma_1, \Gamma_2) = \text{free}(\Gamma_1) \cap \text{free}(\Gamma_2)$
  s.t. $S \cup \Gamma_i$ are T_i-satisfiable for all i,
  then Γ is **satisfiable**.
- Otherwise, **unsatisfiable**.

Nelson-Oppen Method: Overview

Consider quantifier-free conjunctive $(\Sigma_1 \cup \Sigma_2)$-formula F.

Two versions:

- nondeterministic --- simple to present, but high complexity
- deterministic --- efficient

Nelson-Oppen (N-O) method proceeds in two steps:

- Phase 1 (variable abstraction)
  --- same for both versions
- Phase 2
  nondeterministic: guess equalities/disequalities and check
  deterministic: generate equalities/disequalities by equality propagation

Phase 1: Variable abstraction

Given quantifier-free conjunctive $(\Sigma_1 \cup \Sigma_2)$-formula F.
Transform F into two quantifier-free conjunctive formulae

$$\Sigma_1\text{-formula } F_1 \qquad \text{and} \qquad \Sigma_2\text{-formula } F_2$$

s.t. F is $(T_1 \cup T_2)$-satisfiable iff $F_1 \wedge F_2$ is $(T_1 \cup T_2)$-satisfiable
F_1 and F_2 are linked via a set of shared variables.

For term t, let $\mathrm{hd}(t)$ be the root symbol, e.g. $\mathrm{hd}(f(x)) = f$.

Generation of F_1 and F_2

For $i, j \in \{1, 2\}$ and $i \neq j$, repeat the transformations

(1) if function $f \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[f(t_1, \ldots, t, \ldots, t_n)] \quad \Rightarrow \quad F[f(t_1, \ldots, w, \ldots, t_n)] \land w = t$$

(2) if predicate $p \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[p(t_1, \ldots, t, \ldots, t_n)] \quad \Rightarrow \quad F[p(t_1, \ldots, w, \ldots, t_n)] \land w = t$$

(3) if $\mathrm{hd}(s) \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[s = t] \quad \Rightarrow \quad F[\top] \land w = s \land w = t$$

(4) if $\mathrm{hd}(s) \in \Sigma_i$ and $\mathrm{hd}(t) \in \Sigma_j$,

$$F[s \neq t] \quad \Rightarrow \quad F[w_1 \neq w_2] \land w_1 = s \land w_2 = t$$

where w, w_1, and w_2 are fresh variables.

Phase 2: Guess and Check

- Phase 1 separated $(\Sigma_1 \cup \Sigma_2)$-formula F into two formulae:
  Σ_1-formula F_1 and Σ_2-formula F_2
- F_1 and F_2 are linked by a set of shared variables:
  $V = \text{shared}(F_1, F_2) = \text{free}(F_1) \cap \text{free}(F_2)$
- Let E be an equivalence relation over V.
- The arrangement $\alpha(V, E)$ of V induced by E is:
  $$\alpha(V, E) : \bigwedge_{u,v \in V.\ uEv} u = v \ \wedge \bigwedge_{u,v \in V.\ \neg(uEv)} u \neq v$$

Then,
the original formula F is $(T_1 \cup T_2)$-satisfiable iff
there exists an equivalence relation E of V s.t.
(1) $F_1 \wedge \alpha(V, E)$ is T_1-satisfiable, and
(2) $F_2 \wedge \alpha(V, E)$ is T_2-satisfiable.
Otherwise, F is $(T_1 \cup T_2)$-unsatisfiable.

Practical Efficiency

Phase 2 was formulated as "guess and check":
First, guess an equivalence relation E,
then check the induced arrangement.

The number of equivalence relations grows super-exponentially with the # of shared variables. It is given by Bell numbers.
e.g., 12 shared variables $\Rightarrow$ over four million equivalence relations.

Solution: Deterministic Version
Phase 1 as before
Phase 2 asks the decision procedures P_1 and P_2 to propagate new equalities.

Convex Theories

Equality propagation is a decision procedure for convex theories.

Def. A Σ-theory T is convex iff

for every quantifier-free conjunction Σ-formula F

and for every disjunction $\bigvee_{i=1}^{n}(u_i = v_i)$

if $F \vDash \bigvee_{i=1}^{n}(u_i = v_i)$

then $F \vDash u_i = v_i$, for some $i \in \{1, \ldots, n\}$

Convex Theories

- $T_E, T_{\mathbb{R}}, T_{\mathbb{Q}}, T_{\text{cons}}$ are convex
- $T_{\mathbb{Z}}, T_{\mathsf{A}}$ are not convex

Example: $T_{\mathbb{Z}}$ is not convex

Consider quantifier-free conjunction

$$F: \quad 1 \leq z \ \wedge\ z \leq 2 \ \wedge\ u = 1 \ \wedge\ v = 2$$

Then

$$F \ \models\ z = u \vee z = v$$

but

$$F \ \not\models\ z = u$$

$$F \ \not\models\ z = v$$

Example:

The theory of arrays T_A is not convex.
Consider the quantifier-free conjunctive Σ_A-formula

$$F : \; a\langle i \triangleleft v\rangle[j] = v \; .$$

Then

$$F \;\Rightarrow\; i = j \;\vee\; a[j] = v \; ,$$

but

$$F \not\Rightarrow i = j$$
$$F \not\Rightarrow a[j] = v \; .$$

What if T is Not Convex?

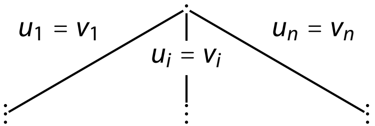
Case split when:

$$\Gamma \models \bigvee_{i=1}^{n} (u_i = v_i)$$

but

$$\Gamma \not\models u_i = v_i \qquad\qquad \text{for all } i = 1, \ldots, n$$

- For each $i = 1, \ldots, n$, construct a branch on which $u_i = v_i$ is assumed.
- If all branches are contradictory, then **unsatisfiable**. Otherwise, **satisfiable**.



Invariant Generation

Invariant Generation

Discover inductive assertions of programs

- General procedure
- Concrete analysis

  - interval analysis

    invariants of form

    $$c \leq v \text{ or } v \leq c$$

    for program variable v and constant c

  - Karr's analysis

    invariants of form

    $$c_0 + c_1 x_1 + \cdots + c_n x_n = 0$$

    for program variables x_i and constants c_i

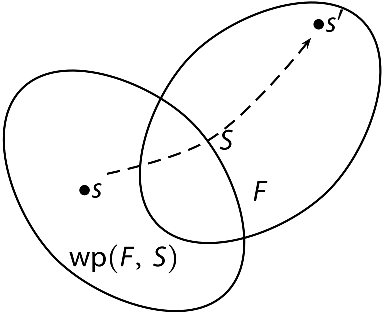
Other invariant generation algorithms in literature:

- linear inequalities

  $c_0 + c_1 x_1 + \cdots + c_n x_n \leq 0$

- polynomial equalities and inequalities

Weakest Precondition



For FOL formula F and program statement S, the weakest precondition $\text{wp}(F,\ S)$ is a FOL formula s.t. if for state s

$$s \models \text{wp}(F,\ S)$$

and if statement S is executed on state s to produce state s', then

$$s' \models F\ .$$

In other words, the weakest precondition moves a formula backwards over a series of statements:
for F to hold after executing $S_1; \ldots; S_n$,
$\mathrm{wp}(F,\ S_1; \ldots; S_n)$ must hold before executing the statements.

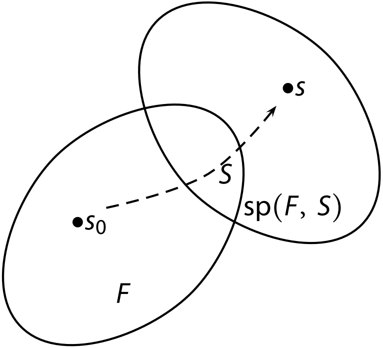
For assume and assignment statements

- $\mathrm{wp}(F,\ \texttt{assume}\ c) \Leftrightarrow c \rightarrow F$, and
- $\mathrm{wp}(F[v],\ v := e) \Leftrightarrow F[e]$;

and on sequences of statements $S_1; \ldots; S_n$:

$$\mathrm{wp}(F,\ S_1; \ldots; S_n) \Leftrightarrow \mathrm{wp}(\mathrm{wp}(F,\ S_n),\ S_1; \ldots; S_{n-1})\ .$$

Strongest Postcondition



For FOL formula F and program statement S, the strongest postcondition sp$(F,\ S)$ is a FOL formula s.t.
if s is the current state and

$$s \models \text{sp}(F,\ S)$$

then statement S was executed from a state s_0 s.t.

$$s_0 \models F\,.$$

- On assume statements,

$$\text{sp}(F,\ \texttt{assume}\ c) \ \Leftrightarrow\ c \ \wedge\ F\ ,$$

  for if program control makes it past the statement, then c must hold.
- Unlike in the case of wp, there is no simple definition of sp on assignments:

$$\text{sp}(F[v],\ v := e[v]) \ \Leftrightarrow\ \exists v^0.\ v = e[v^0] \ \wedge\ F[v^0]\ .$$

- On a sequence of statements $S_1; \ldots; S_n$:

$$\text{sp}(F,\ S_1; \ldots; S_n) \ \Leftrightarrow\ \text{sp}(\text{sp}(F,\ S_1),\ S_2; \ldots; S_n)\ .$$

Example: Compute

$$\begin{aligned} &\mathrm{sp}(i \geq n,\ i := i + k) \\ &\qquad \Leftrightarrow\ \exists i^0.\, i = i^0 + k\ \wedge\ i^0 \geq n \\ &\qquad \Leftrightarrow\ i - k \geq n \end{aligned}$$

since $i^0 = i - k$.

Example: Compute

$$\begin{aligned} &\mathrm{sp}(i \geq n,\ \texttt{assume}\ k \geq 0;\ i := i + k) \\ &\qquad \Leftrightarrow\ \mathrm{sp}(\mathrm{sp}(i \geq n,\ \texttt{assume}\ k \geq 0),\ i := i + k) \\ &\qquad \Leftrightarrow\ \mathrm{sp}(k \geq 0\ \wedge\ i \geq n,\ i := i + k) \\ &\qquad \Leftrightarrow\ \exists i^0.\, i = i^0 + k\ \wedge\ k \geq 0\ \wedge\ i^0 \geq n \\ &\qquad \Leftrightarrow\ k \geq 0\ \wedge\ i - k \geq n \end{aligned}$$

Verification Condition

VCs in terms of wp:

$$\{F\}S_1;\ldots;S_n\{G\}:\ F \ \Rightarrow\ \mathrm{wp}(G,\ S_1;\ldots;S_n)\,.$$

VCs in terms of sp:

$$\{F\}S_1;\ldots;S_n\{G\}:\ \mathrm{sp}(F,\ S_1;\ldots;S_n)\ \Rightarrow\ G\,.$$

Static Analysis: basic definition

- Program P with locations $\mathcal{L}$ (L_0 --- initial location)
- Cutset of $\mathcal{L}$
  each path from one cutpoint (location in the cutset) to the next cutpoint is basic path (does not cross loops)
- Assertion map
  $$\mu : \mathcal{L} \to \text{FOL}$$
  (map from $\mathcal{L}$ to first-order assertions).
  It is inductive (inductive map) if for each basic path

$(\cdot)$

$$\begin{array}{l} L_i : \ @\ \mu(L_i) \\ S_i; \\ \ \vdots \\ S_j; \\ L_j : \ @\ \mu(L_j) \end{array}$$

---

for $L_i, L_j \in \mathcal{L}$, the verification condition

$$\{\mu(L_i)\}S_i; \ldots ; S_j\{\mu(L_j)\} \qquad \text{(VC)}$$

is valid.

Invariant Generation

Find inductive assertion maps μ s.t. the $\mu(L_i)$ satisfies (VC) for all basic paths.

Method: Symbolic execution (forward propagation)

- Initial map μ_0:

$$\begin{array}{ll} \mu(L_0) := F_{\text{pre}}\,, & \text{and} \\ \mu(L) := \bot & \text{for} \quad L \in \mathcal{L} \setminus \{L_0\}. \end{array}$$

- Maintain set $S \subseteq \mathcal{L}$ of locations that still need processing. Initially, let $S = \{L_0\}$. Terminate when $S = \emptyset$.
- Iteration i: We have so far constructed μ_i. Choose some $L_j \in S$ to process and remove it from S.

For each basic path (starting at L_j)

$$(\cdot)$$

$$
\begin{array}{l}
L_j : \ @\ \mu(L_j) \\
S_j; \\
\ \vdots \\
S_k; \\
L_k : \ @\ \mu(L_k)
\end{array}
$$

compute and set

$$\mu(L_k) \ \Leftrightarrow\ \mu(L_k) \vee \mathrm{sp}(\mu(L_j),\ S_j; \ldots ; S_k)$$

If

$$\mathrm{sp}(\mu(L_j),\ S_j; \ldots ; S_k) \ \Rightarrow\ \mu_i(L_k)$$

that is, if *sp* does not represent new states not already represented in $\mu_i(L_k)$, then $\mu_{i+1}(L_k) \ \Leftrightarrow\ \mu_i(L_k)$ (nothing new is learned)

Otherwise add L_k to S.

For all other locations $L_\ell \in \mathcal{L}$, $\mu_{i+1}(L_\ell) \ \Leftrightarrow\ \mu_i(L_\ell)$

When $S = \varnothing$ (say iteration i^*), then μ_{i^*} is an inductive map.

The algorithm

$$
\begin{array}{l}
\texttt{let}\ \text{ForwardPropagate}\ F_{\text{pre}}\ \mathcal{L} = \\
\quad S := \{L_0\}; \\
\quad \mu(L_0) := F_{\text{pre}}; \\
\quad \mu(L) := \bot\ \texttt{for}\ L \in \mathcal{L} \setminus \{L_0\}; \\
\quad \texttt{while}\ S \neq \varnothing\ \texttt{do} \\
\quad\quad \texttt{let}\ L_j\ =\ \text{choose}\ S\ \texttt{in} \\
\quad\quad S := S \setminus \{L_j\}; \\
\quad\quad \texttt{foreach}\ L_k \in \text{succ}(L_j)\ \texttt{do} \left[\begin{array}{l} L_k \in \text{succ}(L_j) \text{ is a } \textbf{successor} \text{ of } L_j \\ \text{if there is a basic path from } L_j \text{ to } L_k \end{array}\right] \\
\quad\quad\quad \texttt{let}\ F\ =\ \text{sp}(\mu(L_j),\ S_j; \ldots; S_k)\ \texttt{in} \\
\quad\quad\quad \texttt{if}\ F \not\Rightarrow \mu(L_k) \\
\quad\quad\quad \texttt{then}\ \mu(L_k) := \mu(L_k)\ \vee\ F; \\
\quad\quad\quad\quad\quad S := S \cup \{L_k\}; \\
\quad\quad \texttt{done}; \\
\quad \texttt{done}; \\
\quad \mu
\end{array}
$$

Problem: algorithm may not terminate

Example: Consider loop with integer variables i and n:

$@L_0 : i = 0 \ \wedge \ n \geq 0;$

`while`

$@L_1 : ?$

$(i < n)\ \{$

$i := i + 1;$

$\}$

There are two basic paths:

(1)

$@L_0 : \ i = 0 \ \wedge \ n \geq 0;$

$@L_1 : ?;$

and

(2)

$@L_1 : ?;$

`assume` $i < n;$

$i := i + 1;$

$@L_1 : ?;$

- Initially,

$$\boxed{\begin{array}{lcl} \mu(L_0) & \Leftrightarrow & i = 0 \ \wedge \ n \geq 0 \\ \mu(L_1) & \Leftrightarrow & \perp \end{array}}$$

- Following path **(1)** results in setting

$$\mu(L_1) := \mu(L_1) \ \vee \ (i = 0 \ \wedge \ n \geq 0)$$

$\mu(L_1)$ was $\perp$, so that it becomes

$$\boxed{\mu(L_1) \ \Leftrightarrow \ i = 0 \ \wedge \ n \geq 0 \, .}$$

- On the next iteration, following path **(2)** yields

$$\mu(L_1) := \mu(L_1) \ \vee \ \mathrm{sp}(\mu(L_1), \ \texttt{assume}\ i < n; \ i := i + 1) \, .$$

Currently $\mu(L_1) \ \Leftrightarrow \ i = 0 \ \wedge \ n \geq 0$, so

$$\begin{aligned} F: \ \mathrm{sp}(i = 0 \ \wedge \ n \geq 0, \ \texttt{assume}\ i < n; \ i := i + 1) \\ \Leftrightarrow \ \mathrm{sp}(i < n \ \wedge \ i = 0 \ \wedge \ n \geq 0, \ i := i + 1) \\ \Leftrightarrow \ \exists i^0 . \, i = i^0 + 1 \ \wedge \ i^0 < n \ \wedge \ i^0 = 0 \ \wedge \ n \geq 0 \\ \Leftrightarrow \ i = 1 \ \wedge \ n > 0 \end{aligned}$$

Since the implication

$$\underbrace{i = 1 \ \wedge \ n > 0}_{F} \ \Rightarrow \ \underbrace{i = 0 \ \wedge \ n \geq 0}_{\mu(L_1)}$$

is invalid,

$$\boxed{\mu(L_1) \ \Leftrightarrow \ \underbrace{(i = 0 \ \wedge \ n \geq 0)}_{\mu(L_1)} \vee \underbrace{(i = 1 \ \wedge \ n > 0)}_{F}}$$

at the end of the iteration.

- At the end of the next iteration,

$$\boxed{\mu(L_1) \ \Leftrightarrow \ \begin{array}{l} \underbrace{(i = 0 \ \wedge \ n \geq 0) \ \vee \ (i = 1 \ \wedge \ n > 0)}_{\mu(L_1)} \\ \vee \ \underbrace{(i = 2 \ \wedge \ n > 1)}_{F} \end{array}}$$

- At the end of the kth iteration,

$$\boxed{\mu(L_1) \iff \begin{array}{l}(i = 0 \ \wedge \ n \geq 0) \ \vee \ (i = 1 \ \wedge \ n \geq 1) \\ \ \vee \ \cdots \ \vee \ (i = k \ \wedge \ n \geq k)\end{array}}$$

  It is never the case that the implication

$$\begin{gathered} i = k \ \wedge \ n \geq k \\ \Downarrow \\ (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \cdots \vee (i = k-1 \wedge n \geq k-1) \end{gathered}$$

  is valid, so the main loop of while never finishes.
- However, it is obvious that

$$0 \leq i \leq n$$

  is an inductive annotation of the loop.

Solution: Abstraction

A state *s* is reachable for program *P* if it appears in some computation of *P*.

The problem is that ForwardPropagate computes the exact set of reachable states.

Inductive annotations usually over-approximate the set of reachable states: every reachable state *s* satisfies the annotation, but other unreachable states can also satisfy the annotation.

Abstract interpretation cleverly over-approximate the reachable state set to guarantee termination.

Abstract interpretation is constructed in 6 steps.