An Interpolating Decision Procedure for Transitive Relations with Uninterpreted Functions

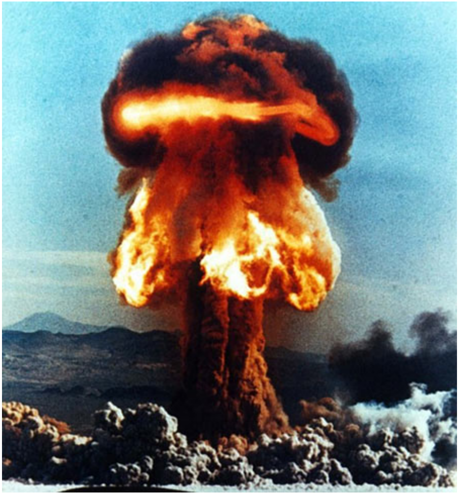
Georg Weissenbacher (白傑岳)

University of Oxford and ETH Zurich

UNU IIST, Macau, 6^{th} of January, 2010

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Motivation



Prevent *bad things* from happening

A certain bad thing is not supposed to happen

$\equiv$

`assert`($\neg$bad thing)

$\equiv$

safety / reachability property

Safety Properties, Assertions

A certain bad thing is not supposed to happen

$$\equiv$$

`assert`($\neg$bad thing)

$$\equiv$$

safety / reachability property

Assertions:

- Supported by main-stream languages such as ANSI-C, C++, Java
- Widely accepted by programmers
- Easy to generate (buffer overflows, division by 0, etc.)

A certain bad thing is not supposed to happen

$\equiv$

`assert`($\neg$bad thing)

$\equiv$

safety / reachability property

Assertions:

- Supported by main-stream languages such as ANSI-C, C++, Java
- Widely accepted by programmers
- Easy to generate (buffer overflows, division by 0, etc.)

Prove safety of program or find counterexample using Model Checking

Outline

- Part I: Interpolant-based model checking
  - Background (predicate transformers, interpolants, safety invariants)
  - Example
- Part II: An interpolating decision procedure
  - A proof-generating decision procedure
  - Deriving interpolants from proofs

Hoare Triples

- Program assertions represented by predicates
- $\{P\}\ \texttt{instruction}\ \{Q\}$

  "if P holds, Q will hold after `instruction` terminates"
- Example of a Hoare rule:

$$\frac{}{\{P[\mathrm{x}/\mathit{expr}]\}\ \texttt{x:=}\mathit{expr}\ \{P\}}\ \text{assignment}$$

Hoare Triples

- Program assertions represented by predicates
- $\{P\}$ `instruction` $\{Q\}$

  "if P holds, Q will hold after `instruction` terminates"
- Example of a Hoare rule:

$$\frac{}{\{P[\mathtt{x}/expr]\}\ \mathtt{x}{:=}expr\ \{P\}}\ \text{assignment}$$

- Alternative view: Instructions represented by predicates

$$\begin{array}{ccccc} P(x) & \wedge & T(x,x') & \Rightarrow & Q(x') \\ (x=5) & \wedge & (x'=x+1) & \Rightarrow & (x'\neq 5) \end{array}$$

- Strongest post-condition:

$$\{P\}\ \texttt{x := expr;}\ \{\texttt{Q}\} \qquad Q \equiv (\exists x\,.P \wedge x' = \text{expr})$$
$$\{P\}\ [\texttt{expr}]\ \{\texttt{Q}\} \qquad Q \equiv P \wedge \text{expr}$$

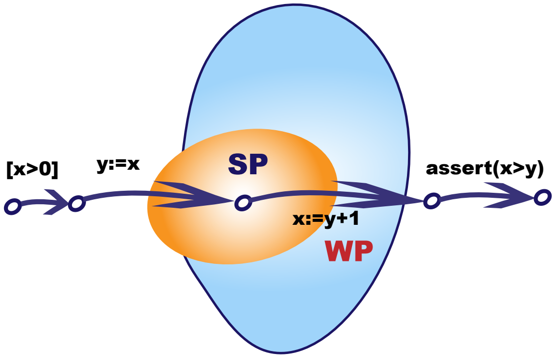
- Weakest pre-condition:

$$\{P\}\ \texttt{x := expr;}\ \{\texttt{Q}\} \qquad P \equiv Q[x/\text{expr}]$$
$$\{P\}\ [\texttt{expr}]\ \{\texttt{Q}\} \qquad P \equiv \text{expr} \Rightarrow Q$$

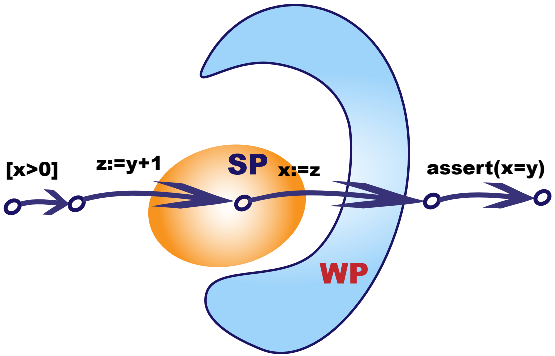
- Composition rule for two sub-paths π_1 and π_2:

$$\frac{\{P\}\ \pi_1\ \{Q\},\ \ \{Q\}\ \pi_2\ \{R\}}{\{P\}\ \pi_1\,;\pi_2\ \{R\}}\ \text{composition}$$

- Loops: *Fixed-point* computation (cf. Dijkstra)
  "good invariants" are hard to find

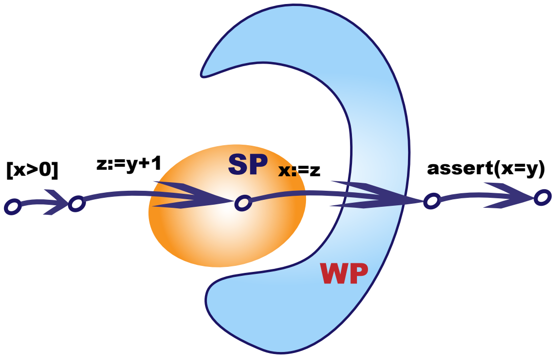


SP: $(x > 0) \land y = x$ WP: $y + 1 > y$



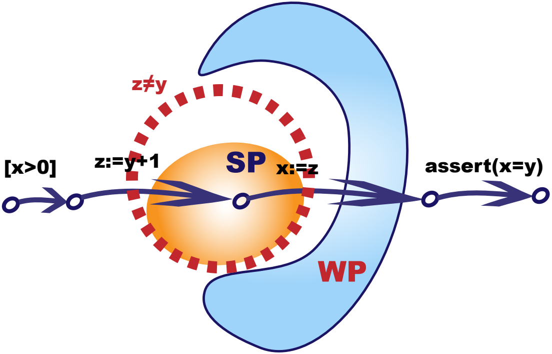
SP: $(x > 0) \wedge z = y + 1$ WP: $z = y$

Infeasible paths



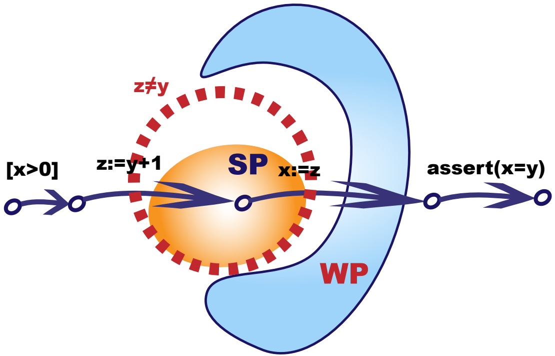
SP: $(x > 0) \wedge z = y + 1$ WP: $z = y$ $(z = y + 1) \wedge (z = y) \Rightarrow \text{false}$

Infeasible paths (continued)



SP: $(x > 0) \wedge z = y + 1$

WP: $z = y$

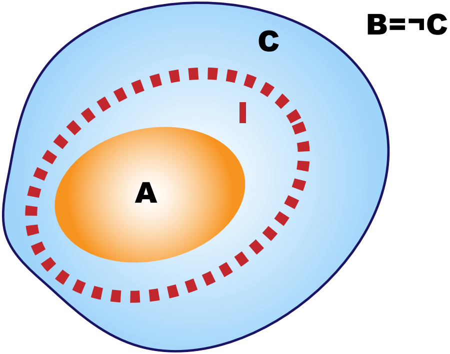


SP: $(x > 0) \wedge z = y + 1 \Rightarrow z \neq y$ WP: $z = y$

What is a Craig interpolant?

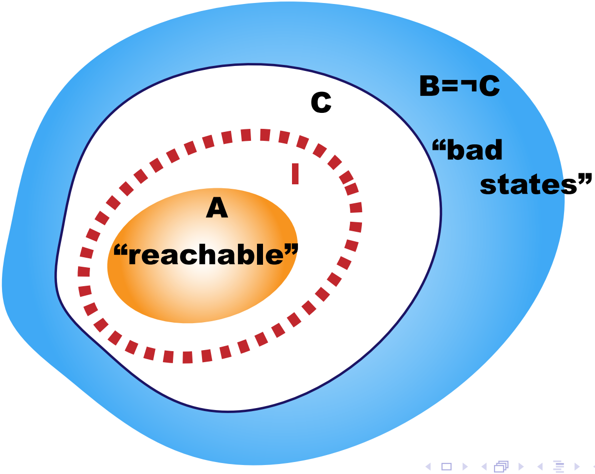
"Traditional" definition [William Craig, 57]:

- $A \Rightarrow I \Rightarrow C$
- all non-logical symbols in I occur in A *as well as* in C



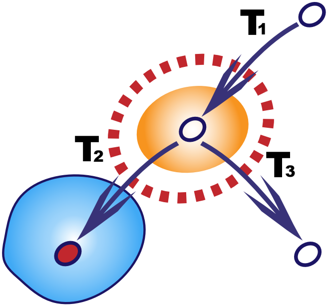
Common definition for automated verification:

- $A \Rightarrow I$ and $I \wedge B$ inconsistent
- all non-logical symbols in I occur in A *as well as* in B



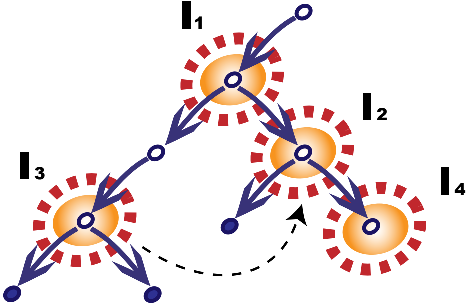
Over-approximation of reachable *safe states* in a program:

- T_ℓ: transition function for each location $\ell \in \{1, 2, 3, \ldots\}$
- $T_1(x_1, x_2) \wedge T_2(x_2, x_3)$ symbolic representation of (infeasible) path
- $T_1(x_1, x_2) \Rightarrow I(x_2) \quad I(x_2) \wedge T_2(x_2, x_3)$ inconsistent



Safety Invariant, Covered Nodes

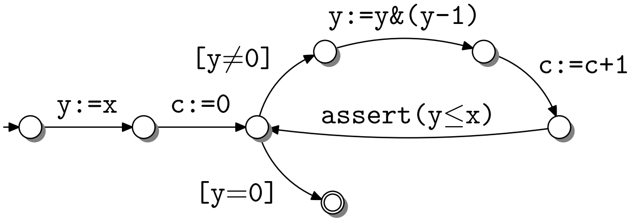
- Safety Invariant: $I \wedge T \Rightarrow I'$ and "bad" locations are labelled "**false**"
- If $I_3 \Rightarrow I_2$ then the node labelled "I_3" and its successors are covered



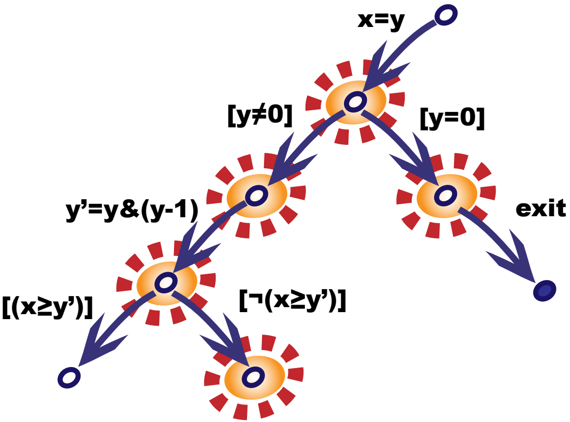
A small example: Wegner's bit-counting algorithm.

```
y:=x; c:=0;
while (y ≠ 0) {
   y:=y & (y-1);
   c:=c+1;
   assert(x ≥ y);
}
```

Representation as control flow graph (CFG):



Unwinding the loop once. . .

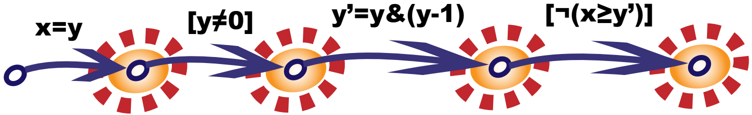


An inconsistent formula representing an infeasible path

$$(x = y) \wedge (y \neq 0) \wedge (y' = y\&(y-1)) \wedge (\neg(x \geq y'))$$

Step	SP	ITP	$\neg$WP
1	$x = y$	$x = y$	$(x \geq y\&(y-1)) \vee (y = 0)$
2			
3			
4			

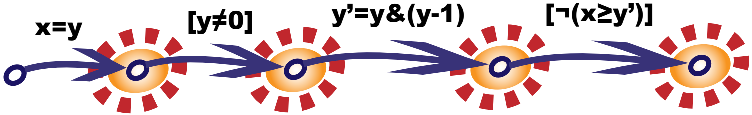
An inconsistent formula representing an infeasible path



$$(x = y) \wedge (y \neq 0) \wedge (y' = y\&(y-1)) \wedge (\neg(x \geq y'))$$

Step	SP	ITP	¬WP
1	$x = y$	$x = y$	$(x \geq y\&(y-1)) \vee (y = 0)$
2	$x = y \wedge y \neq 0$	$x = y$	$(x \geq y\&(y-1))$
3			
4			

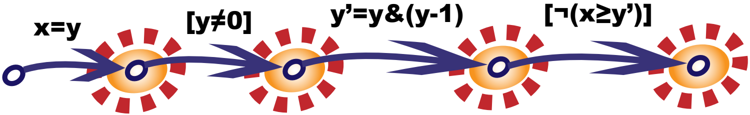
An inconsistent formula representing an infeasible path



$$(x = y) \wedge (y \neq 0) \wedge (y' = y\&(y-1)) \wedge (\neg(x \geq y'))$$

Step	SP	ITP	$\neg$WP
1	$x = y$	$x = y$	$(x \geq y\&(y-1)) \vee (y = 0)$
2	$x = y \wedge y \neq 0$	$x = y$	$(x \geq y\&(y-1))$
3	$y' = x\&(x-1) \wedge x \neq 0$	$x \geq y'$	$x \geq y'$
4			

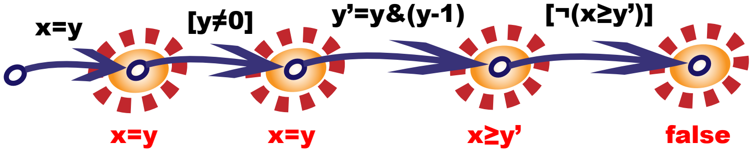
An inconsistent formula representing an infeasible path



$$(x = y) \wedge (y \neq 0) \wedge (y' = y\&(y-1)) \wedge (\neg(x \geq y'))$$

Step	SP	ITP	$\neg$WP
1	$x = y$	$x = y$	$(x \geq y\&(y-1)) \vee (y = 0)$
2	$x = y \wedge y \neq 0$	$x = y$	$(x \geq y\&(y-1))$
3	$y' = x\&(x-1) \wedge x \neq 0$	$x \geq y'$	$x \geq y'$
4	false	false	false

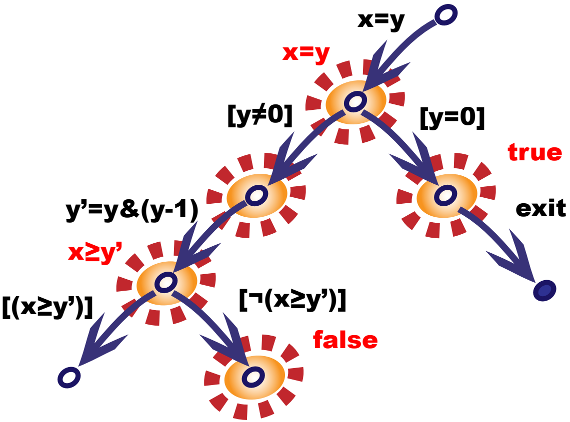
An inconsistent formula representing an infeasible path

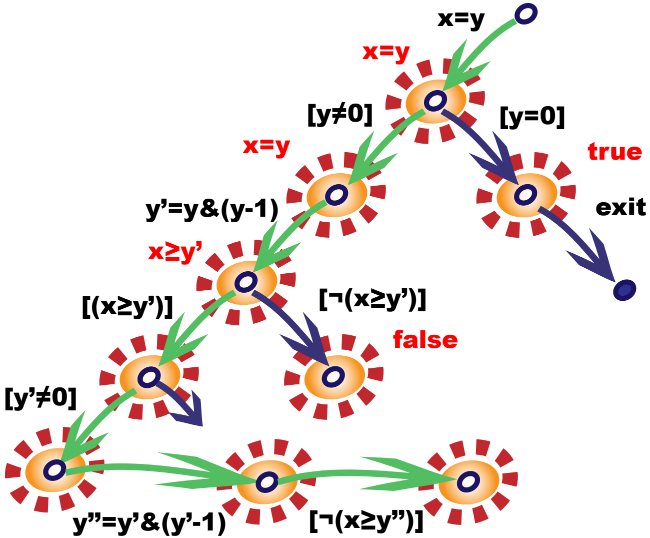


$$(x = y) \wedge (y \neq 0) \wedge (y' = y\&(y-1)) \wedge (\neg(x \geq y'))$$

Step	SP	ITP	$\neg$WP
1	$x = y$	$x = y$	$(x \geq y\&(y-1)) \vee (y = 0)$
2	$x = y \wedge y \neq 0$	$x = y$	$(x \geq y\&(y-1))$
3	$y' = x\&(x-1) \wedge x \neq 0$	$x \geq y'$	$x \geq y'$
4	false	false	false

A small example (now with interpolants)




x=y
x=y
[y≠0]
[y=0]
x=y
true
y'=y&(y-1)
exit
x≥y'
[(x≥y')]
[¬(x≥y')]
false
[y'≠0]
y''=y'&(y'-1)
[¬(x≥y'')]

Path prefix:

$$(x = y) \wedge (y \neq 0) \wedge (y' = y \& (y-1)) \wedge (x \geq y') \wedge (y' \neq 0) \wedge (y'' = y' \& (y'-1))$$

Assertion:

$$\neg(x \geq y'')$$

Interpolant:

$$x \geq y''$$

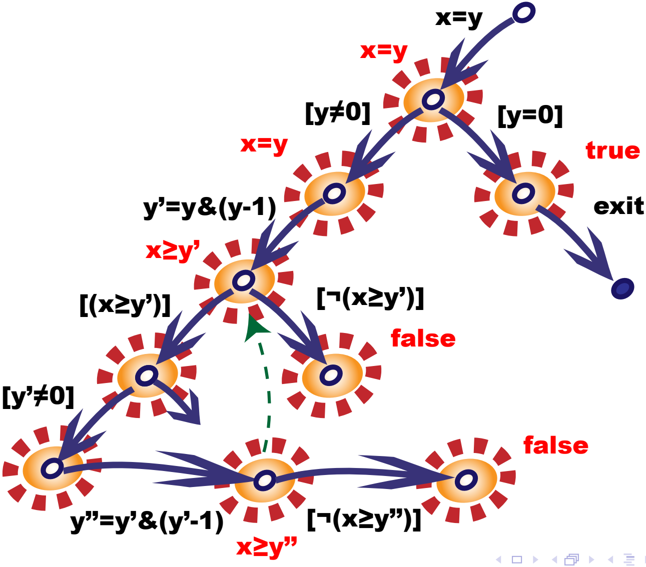
follows from:

$$x \geq y',\ (y'' = y' \& (y' - 1)) \text{ implies } (y' \geq y''), \text{ and transitivity}$$

Strongest post-condition: (by means of substitution)

$$y'' = (x \& (x-1)) \,\&\, ((x \& (x-1)) - 1) \wedge (x \neq 0) \wedge (x \,\&\, (x-1) \neq 0)$$

Succeeded to prove safety!



- Given a sequence of transitions $T_0 \wedge T_1 \wedge \ldots \wedge T_n$
- let I_i be the interpolant for

$$T_0 \wedge T_1 \wedge \ldots \wedge T_{i-1} \qquad \text{and} \qquad T_i \wedge \ldots \wedge T_{n-1} \wedge T_n$$

- then it has to hold that

$$\begin{aligned} I_0 &= \textbf{true} \\ I_{n+1} &= \textbf{false} \\ \forall i \in \{1, n\} \,.\, I_i \wedge T_i &\Rightarrow I_{i+1} \end{aligned}$$

How is the transition function T encoded?

Currently:

- Boolean connectives
- Equality
- Uninterpreted functions
- Difference logic, linear arithmetic

Problem: Programs have *bit-vector* semantics and bit-vector operations.

$$a > b + 2 \wedge a \leq b$$

- Unsatisfiable in the theory of linear arithmetic ($\mathbb{R}$, $\mathbb{Z}$, ...)

Currently:

- Boolean connectives
- Equality
- Uninterpreted functions
- Difference logic, linear arithmetic

Problem: Programs have *bit-vector* semantics and bit-vector operations.

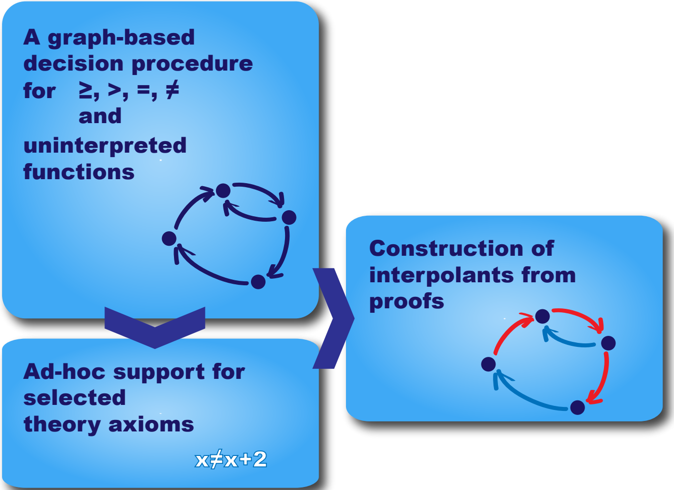
$$a > b + 2 \wedge a \leq b \qquad \{a \mapsto 2, b \mapsto 2\}$$

- Unsatisfiable in the theory of linear arithmetic ($\mathbb{R}$, $\mathbb{Z}$, $\ldots$)
- Satisfiable if *a* and *b* are **2-bit bit-vectors**

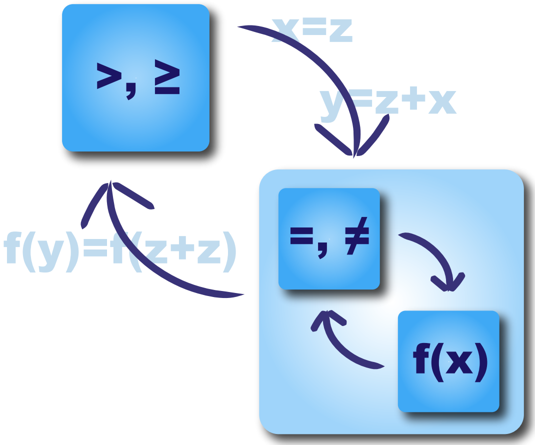
Proposed solution

- Provide proof-generating decision procedure for *conjunctions* of
  - Strict and weak inequalities ($<, \leq$)
  - Equalities and dis-equalities ($=, \neq$)
  - both with uninterpreted functions (UF)
- Deal with theory specific terms in an ad-hoc manner
  - Constant propagation
  - Simplify ground terms (bit-level accurate)
  - *Limited* application of theory axioms

Propositional structure can be dealt with using SMT and [Yorsh + Musuvathi, 05]

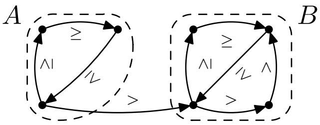

A graph-based decision procedure for ≥, >, =, ≠ and uninterpreted functions
Ad-hoc support for selected theory axioms
x≠x+2
Construction of interpolants from proofs

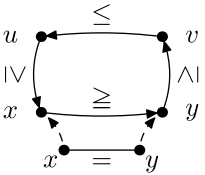
Overview: Proof-generating Decision Procedure



Weak and strong inequalities

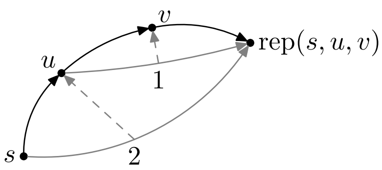
- Add all facts $s < t$ and $s \leq t$ to directed graph $\mathcal{G}$
- Compute **Strongly Connected Components** (SCCs)

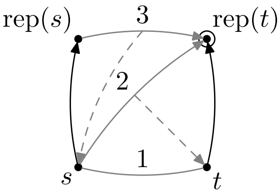




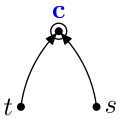
- If SCC contains an edge $s < t$:
  - find shortest path from s to t
  - report contradictory cycle
- Otherwise: For each $s \leq t$ in SCC
  - add $s = t$ as a fact

- Add all facts $s = t$ to graph-based **Union-Find** data structure $\mathcal{U}$
- Modify *Find*-operation / path-compression:
  - remember the 2 edges entailing shortcut
- Modify *Union*-operation:
  - triangulate sub-graph s – rep(s) – rep(t) – t
- Perform query for each $s \neq t$





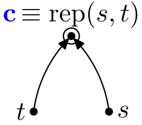
- Proof-producing congruence closure [Nieuwenhuis, Oliveras 05]
- Observation:



$$\Rightarrow \qquad f(t) = f(s)$$

Uninterpreted Functions (continued)

- Based on Union-Find data structure $\mathcal{U}$:
  - Maintain a `use_list` of encountered terms $f(t)$ that "use" c



$$\texttt{use\_list}[\mathbf{c}] = [\,\mathrm{f(t)}, \mathrm{g(s)}, \ldots\,]$$

  - For each $f(c)$

$$\texttt{lookup}(f, c) = \begin{cases} f(t) & \text{an element which maps to } f(c) \\ \perp & \textit{otherwise} \end{cases}$$

- If the representative constant c changes to c'
  - For all $f(t) \in \texttt{use\_list}[c]$:

$$
\begin{array}{ll}
\text{add } (f(t) = f(s)) \text{ to } \mathcal{U} & \text{if } \texttt{lookup}(f, c') = f(s) \\
\texttt{lookup}(f, c') \stackrel{\textit{def}}{=} f(t) & \text{if } \texttt{lookup}(f, c') = \bot
\end{array}
$$

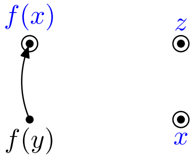
Update `use_list` accordingly.

- If the representative constant c changes to c'
  - For all $f(t) \in \texttt{use\_list}[c]$:

$$\begin{array}{ll} \text{add } (f(t) = f(s)) \text{ to } \mathcal{U} & \text{if } \texttt{lookup}(f, c') = f(s) \\ \texttt{lookup}(f, c') \stackrel{def}{=} f(t) & \text{if } \texttt{lookup}(f, c') = \perp \end{array}$$

  Update use_list accordingly.

- Example:



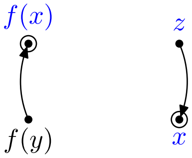
$\texttt{use\_list}[z] = [\, f(z) \,]$

- If the representative constant c changes to c'
  - For all $f(t) \in \texttt{use\_list}[c]$:

$$\begin{array}{ll} \text{add } (f(t) = f(s)) \text{ to } \mathcal{U} & \text{if } \texttt{lookup}(f, c') = f(s) \\ \texttt{lookup}(f, c') \stackrel{def}{=} f(t) & \text{if } \texttt{lookup}(f, c') = \perp \end{array}$$

  Update use_list accordingly.

- Example:



$\texttt{use\_list}[z] = [\, f(z) \,]$

Uninterpreted Functions (continued)

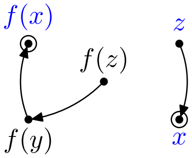
- If the representative constant c changes to c'
  - For all $f(t) \in \texttt{use\_list}[c]$:

$$\text{add } (f(t) = f(s)) \text{ to } \mathcal{U} \quad \text{if } \texttt{lookup}(f, c') = f(s)$$

$$\texttt{lookup}(f, c') \stackrel{def}{=} f(t) \quad \text{if } \texttt{lookup}(f, c') = \perp$$

  Update use_list accordingly.

- Example:



$\texttt{use\_list}[z] = [\quad\quad]$

Constant Propagation and Simplification

- For each equivalence class in $\mathcal{U}$
  - Track *theory-specific* constants (e.g., numerical) in $\mathcal{U}$
  - W.l.o.g., *one* constant per equivalence class (otherwise contradictory)
- For sub-term-closed pool of expressions encountered so far:
  - *substitute* constants for sub-terms
  - *simplify* and add respective equivalence, e.g., $(x\&0) = 0$

- Apply term-rewriting rules, e.g.,

$$\frac{c \neq 0 \bmod 2^m}{(x+c) \neq x} \qquad \frac{c = 0 \bmod 2^m}{(x+c) = x} \qquad \frac{1 \leq c < m}{(t{<<}c) = (2^c \cdot t)}$$

(for m-bit variables x) if respective terms are encountered.

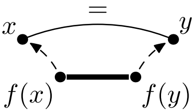
- Apply theory-specific axioms, e.g.,

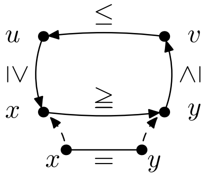
$$\frac{t_1 = t_2 \,\&\, t_3}{t_1 \leq t_2 \quad t_1 \leq t_3} \qquad \frac{t_1 = t_2 \mid t_3}{t_1 \geq t_2 \quad t_1 \geq t_3} \qquad \frac{t_1 + t_2 = t_1}{t_2 = 0}$$

Important: These rules do not introduce non-logical symbols

Derived edges, Proof of inconsistency

- Keep track of premises for inferred equivalences!



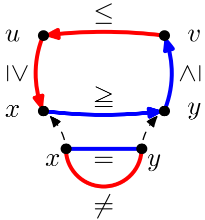


A **proof of inconsistency** consists of

- a contradictory cycle (contains $\leq$ and $<$ *or* $=$ and exactly one $\neq$)
- premises for all derived edges

Interpolation

- *Intuition:* Split **proof** into **facts** contributed by A and B, respectively!



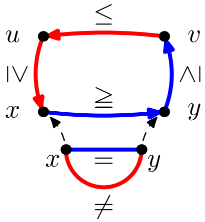
$$(x \geq y) \wedge (y \geq v) \qquad (v \geq u) \wedge (u \geq x) \wedge (x \neq y)$$

- A *fact* is a maximal path in which all edges have the same colour:

  $x \geq v$, $v \geq x$, $x = y$, $x \neq y$

- An **interpolant** can be seen as *assume-guarantee* reasoning:

$$\boxed{A \text{ guarantees } x = y \textit{ iff } B \text{ does not violate } v \geq x}$$



- **Interpolant:**

$$x = y \;\vee\; \neg(v \geq x)$$

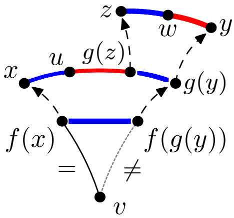
Split the **proof of inconsistency** into two components $\mathcal{I}$ and $\mathcal{J}$:

- $\mathcal{J}$: A set of tuples $\langle P, t = s \rangle$
  - P contains "all A-coloured facts needed to justify $t = s$"
  - $P \subseteq \mathcal{I}$
- $\mathcal{I}$: A set of A-"coloured" facts.
  - $\mathcal{J}$ contains "all B-coloured facts needed to justify $(t = s) \in \mathcal{I}$"

- *B*-**premise** of $(t = s)$:

  "all *B*-coloured facts needed to justify $(t = s)$"

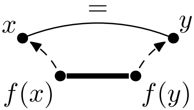
- Example:

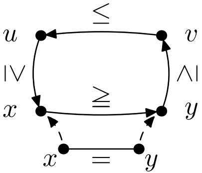


B-**premise**(f(x)=f(g(y))) = $\{u = g(z),\ w = y\}$

- Definition of *A*-**premise** is symmetric

- Conditions:





- Premises:

$$A\text{-premise}\,(v_i \xrightarrow{=} v_j) \stackrel{\text{def}}{=}$$

$$(A\text{-condition for } v_i \xrightarrow{=} v_j) \cup$$

$$\bigcup \{A\text{-premise}\,(v_n \rightarrow v_m) \mid v_n \rightarrow v_m \in (\text{B-condition for } v_i \xrightarrow{=} v_j)\}$$

Interpolant for $\mathcal{I}$ and $\mathcal{J}$

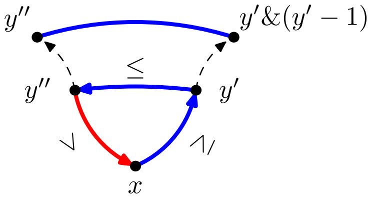
$$I \stackrel{\text{def}}{=} \bigwedge_{v_i \xrightarrow{\triangleright} v_j \in \mathcal{I}} (t_i \triangleright t_j) \vee \underbrace{\bigvee_{\langle P, v_n \xrightarrow{\triangleright} v_m \rangle \in \mathcal{J}} \left(\bigwedge_{(v_i \xrightarrow{\triangleright P} v_j) \in P} (t_i \triangleright_P t_j) \right) \wedge \neg (t_n \triangleright t_m)}_{\text{“challenges” B to “break the contract”}}$$

B can either

- try to pretend that one $\neg(t_n \triangleright t_m)$ holds and contradict itself
- admit that all $(t_n \triangleright t_m)$ hold and contradict $\bigwedge_{v_i \xrightarrow{\triangleright} v_j \in \mathcal{I}} (t_i \triangleright t_j)$

$$(x = y) \wedge (y \neq 0) \wedge (y' = y\&(y-1)) \wedge (x \geq y') \wedge (y' \neq 0) \wedge (y'' = y'\&(y'-1))$$

$$\neg(x \geq y'')$$



Interpolant: $x \geq y'' \vee x \geq y''$

Conclusion

- New interpolating decision procedure
  - *algorithmic* description (vs. axiomatic in [McMillan 05])
  - based on work by [Nieuwenhuis, Oliveras 05], McMillan, [Fuchs, Goel, Grundy, Krstić, Tinelli 09].
- Sound for bit-vector semantics (*not* a bit-vector decision procedure!)
- "Good-enough" philosophy:
  Avoid using a complete decision system for arithmetic in favour of *ad-hoc* treatment of ground terms
  - Implemented interpolation-based model checker WOLVERINE
  - Decision procedure is sufficient for typical Windows device driver examples (`kbfiltr`, `floppy`, `mouclass`, ...)

Outlook

- *Interpolant Strength*
  V. D'Silva, D. Kroening, M. Purandare, G. Weissenbacher
  VMCAI, January 2010, Madrid (co-located with POPL)
- Generating interpolants of different strength wrt. the implication order

Computing $\mathcal{I}$ and $\mathcal{J}$

let $\mathcal{G}(V_A \cap V_B, E_A \cup E_B)$ **be** the factorised and contracted proof
let E_C **be** the facts in the contradictory cycle of $\mathcal{G}$
$\mathcal{W} := E_C$, $\mathcal{I} := \emptyset$, $\mathcal{J} := \emptyset$
while $(\mathcal{W} \neq \emptyset)$ **do**
  remove $v_i \to v_j$ from $\mathcal{W}$
  if $v_i \to v_j$ is B-coloured **then**
    $P := A\text{-premise}(v_i \to v_j)$
    $\mathcal{J} := \mathcal{J} \cup \{\langle P, v_i \to v_j \rangle\}$
  else
    $P := B\text{-premise}(v_i \to v_j)$
    $\mathcal{I} := \mathcal{I} \cup \{v_i \to v_j\}$
  end if
  $\mathcal{W} := \mathcal{W} \cup P$
end while