Program Verification Recap

Christian Schilling

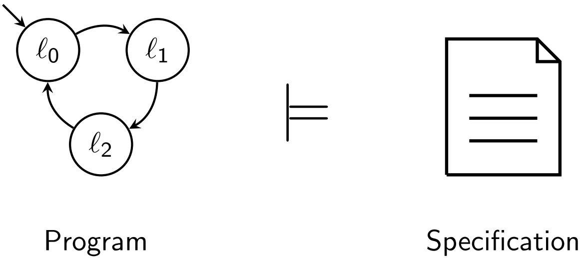
July 25/26, 2017

Overview

What is program verification?

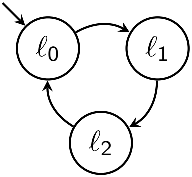
"Da stelle mehr uns janz dumm und da sage mer so . . ."

What is program verification?



What is program verification?

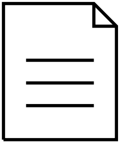
Behavior of



Program

$\subseteq$

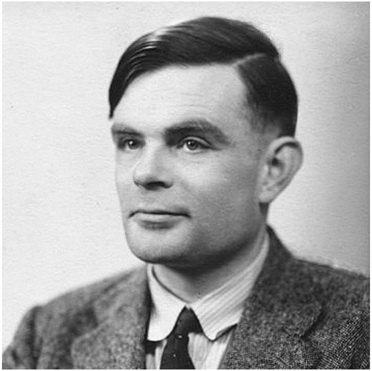
Behavior of



Specification

Milestones

- Alan M. Turing
- Halting problem ’36



Milestones

- Henry G. Rice
- Rice's theorem '51

The question
"Program $\models$ Specification"
is undecidable.

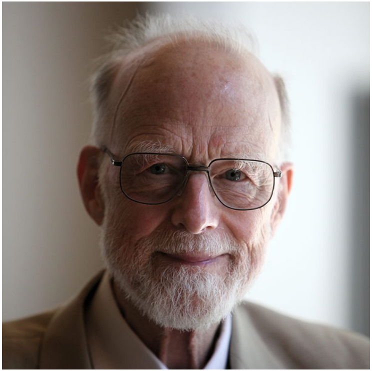
Even worse, it remains undecidable for any fixed specification different from *true* and *false*.

Milestones

- Robert W. Floyd
- Assertions in flow charts '67
- Turing award '78

Milestones

- C.A.R. "Tony" Hoare
- Hoare logic '69
- Turing award '80



Milestones

- Edsger W. Dijkstra
- Guarded commands & weakest precondition '75
- Turing award '72

Milestones

- Patrick & Radhia Cousot
- Abstract interpretation '77

What are programs?

- Semantics
  - Axiomatic (transition = effect on assertions)
  - Operational (transition = set of pairs of states)
  - Denotational (program = mathematical object, e.g., function)

What are programs?

- Semantics
  - Axiomatic (transition = effect on assertions)
  - Operational (transition = set of pairs of states)
  - Denotational (program = mathematical object, e.g., function)
- Different views & aspects

What are programs?

- Semantics
  - Axiomatic (transition = effect on assertions)
  - Operational (transition = set of pairs of states)
  - Denotational (program = mathematical object, e.g., function)
- Different views & aspects
  - Sequence of commands

Commands

$$
\begin{aligned}
C ::= & \ \textbf{skip} \\
& \mid C; C \\
& \mid \mathsf{x} := \mathsf{e} \\
& \mid \textbf{if } (\mathsf{b}) \textbf{ then } C \textbf{ else } C \\
& \mid \textbf{while } (\mathsf{b}) \textbf{ do } C
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{x} &::= x_1 \mid \cdots \mid x_n \\
\mathsf{e} &::= \mathsf{x} \mid f(e, \ldots, e) \\
\mathsf{b} &::= \mathsf{x}_b \mid f_b(e, \ldots, e)
\end{aligned}
$$ [1]

For simplicity we ignore type errors and restrict ourselves to one variable domain, usually the integers $\mathbb{Z}$

Each command is deterministic

[1] Here x_b are Boolean variables and f_b map to the Boolean domain

Guarded commands

$$
\begin{aligned}
C ::= {} & \textbf{skip} \\
& \mid C; C \\
& \mid \textbf{havoc}\ \mathsf{x} \\
& \mid C\ [\!]\ C \\
& \mid \textbf{assume}\ \mathsf{b} \\
& \mid \textbf{assert}\ \mathsf{b}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{x} &::= x_1 \mid \cdots \mid x_n \\
\mathsf{e} &::= \mathsf{x} \mid f(e, \ldots, e) \\
\mathsf{b} &::= \mathsf{x}_b \mid f_b(e, \ldots, e)
\end{aligned}
$$
[1]

For simplicity we ignore type errors and restrict ourselves to one variable domain, usually the integers $\mathbb{Z}$

Guarded commands allow for nondeterminism

[1] Here x_b are Boolean variables and f_b map to the Boolean domain

What are programs?

- Semantics
  - Axiomatic (transition = effect on assertions)
  - Operational (transition = set of pairs of states)
  - Denotational (program = mathematical object, e.g., function)
- Different views & aspects
  - Sequence of commands
  - Program state transformers

Program states

Program states

- Valuation of program variables + program counter

$$s : \mathsf{Var} \rightarrow \mathsf{Val}$$

- Set of states symbolically described by a predicate
  We often mix sets and formulas

Program states

- Valuation of program variables + program counter

$$s : \mathsf{Var} \to \mathsf{Val}$$

- Set of states symbolically described by a predicate
  We often mix sets and formulas

- A command transforms a state to a state

- We can lift the definition to sets of states

  Example:

$$\begin{aligned} \text{old states } S: &\quad x = 0 \land y > 2 \\ \text{command } C: &\quad x := y - x \\ \text{new states } S': &\quad x = y \land y > 2 \\ &\quad \triangleq \{s' \mid (C, s) \rightsquigarrow s', s \in S\} \end{aligned}$$

Predicate transformers

- Forward computation:

$$(C, s) \rightsquigarrow s'$$

Predicate transformers

- Forward computation: Strongest postcondition

$$(C, s) \rightsquigarrow s' \triangleq s' \in post(\{s\}, C)$$

Predicate transformers

- Forward computation: Strongest postcondition

$$(C, s) \rightsquigarrow s' \triangleq s' \in post(\{s\}, C)$$

- Backward computation:

$$(C, s) \rightsquigarrow s'$$

Predicate transformers

- Forward computation: Strongest postcondition

$$(C, s) \rightsquigarrow s' \triangleq s' \in post(\{s\}, C)$$

- Backward computation: Weakest precondition

$$(C, s) \rightsquigarrow s' \triangleq s \in wp(\{s'\}, C)$$

- Connection between *wp* and *post*:

$$\varphi \subseteq wp(\psi, C) \iff \psi \quad post(\varphi, C)$$

Predicate transformers

- Forward computation: Strongest postcondition

$$(C, s) \rightsquigarrow s' \triangleq s' \in post(\{s\}, C)$$

- Backward computation: Weakest precondition

$$(C, s) \rightsquigarrow s' \triangleq s \in wp(\{s'\}, C)$$

- Connection between *wp* and *post*: ($\subseteq$ is the same as $\Longrightarrow$)

$$\varphi \subseteq wp(\psi, C) \qquad \Longleftrightarrow \qquad post(\varphi, C) \subseteq \psi$$

What are programs?

- Semantics
  - Axiomatic (transition = effect on assertions)
  - Operational (transition = set of pairs of states)
  - Denotational (program = mathematical object, e.g., function)
- Different views & aspects
  - Sequence of commands
  - Program state transformers
  - Relations between program states

Program state relations

$$
\begin{aligned}
(C, s) \rightsquigarrow s' &\triangleq s' \in post(\{s\}, C) \\
&\triangleq s \in wp(\{s'\}, C)
\end{aligned}
$$

Program state relations

$$
\begin{aligned}
(C, s) \rightsquigarrow s' &\triangleq s' \in post(\{s\}, C) \\
&\triangleq s \in wp(\{s'\}, C) \\
&\triangleq (s, s') \in \rho
\end{aligned}
$$

Program state relations

$$
\begin{aligned}
(C, s) \leadsto s' &\triangleq s' \in post(\{s\}, C) \\
&\triangleq s \in wp(\{s'\}, C) \\
&\triangleq (s, s') \in \rho
\end{aligned}
$$

- In logical characterization: predicates over unprimed and primed variables

  Example: $x := x + 1$ for variables x and y

Program state relations

$$
\begin{aligned}
(C, s) \leadsto s' &\triangleq s' \in post(\{s\}, C) \\
&\triangleq s \in wp(\{s'\}, C) \\
&\triangleq (s, s') \in \rho
\end{aligned}
$$

- In logical characterization: predicates over unprimed and primed variables

  Example: $x := x + 1$ for variables x and y

  $$\rho = \{(x, y, x', y') \mid x' = x + 1 \wedge y' = y\}$$

  or simply

  $$\rho \equiv x' = x + 1 \wedge y' = y$$

What are specifications?

- Two major types of properties

What are specifications?

- Two major types of properties
  - Safety ("Something bad will never happen")
    Correctness = unreachability of error states

What are specifications?

- Two major types of properties
  - Safety ("Something bad will never happen")
    Correctness = unreachability of error states
  - Liveness ("Something good will eventually happen")
    In this lecture: termination

Infinity

- Can we handle finite state systems?

Infinity

- Can we handle finite state systems?
  - Everything is decidable, but very (really!) hard

Infinity

- Can we handle finite state systems?
  - Everything is decidable, but very (really!) hard
- Can we handle infinite state systems?

Infinity

- Can we handle finite state systems?
  - Everything is decidable, but very (really!) hard
- Can we handle infinite state systems?
  - Everything is undecidable except for special subclasses

Infinity

- Can we handle finite state systems?
  - Everything is decidable, but very (really!) hard
- Can we handle infinite state systems?
  - Everything is undecidable except for special subclasses
  - Key idea: make everything finite ($\rightarrow$ abstraction)

Overview

Hoare logic

- (Partial) Correctness specification given as annotation with precondition and postcondition

$$\{\,\varphi\,\}\,P\,\{\,\psi\,\}$$
$$\iff \text{“assume } \varphi\text{, execute } P\text{, assert } \psi\text{”}$$

Hoare logic

- (Partial) Correctness specification given as annotation with precondition and postcondition

$$
\begin{aligned}
& \{\, \varphi \,\}\, P \,\{\, \psi \,\} \\
\iff & \text{“assume } \varphi\text{, execute } P\text{, assert } \psi\text{”} \\
\iff & \varphi \subseteq wp(\psi, P)
\end{aligned}
$$

Hoare logic

- (Partial) Correctness specification given as annotation with precondition and postcondition

$$\begin{aligned} & \{\,\varphi\,\}\, P \,\{\,\psi\,\} \\ \iff & \text{“assume } \varphi\text{, execute } P\text{, assert } \psi\text{”} \\ \iff & \varphi \subseteq wp(\psi, P) \\ \iff & post(\varphi, P) \subseteq \psi \end{aligned}$$

- Calculus (e.g., *wp*) to automatically derive correctness
  Generates verification conditions

Hoare logic

- (Partial) Correctness specification given as annotation with precondition and postcondition

$$
\begin{aligned}
& \{\,\varphi\,\}\, P \,\{\,\psi\,\} \\
\iff & \text{“assume } \varphi\text{, execute } P\text{, assert } \psi\text{”} \\
\iff & \varphi \subseteq wp(\psi, P) \\
\iff & post(\varphi, P) \subseteq \psi
\end{aligned}
$$

- Calculus (e.g., *wp*) to automatically derive correctness
  Generates verification conditions

- precondition, postcondition

$$\frac{\{\,\varphi\,\}\, P \,\{\,\psi\,\}}{\{\,\varphi'\,\}\, P \,\{\,\psi'\,\}}$$

$\varphi' \to \varphi \wedge \psi \to \psi'$

Hoare logic

- (Partial) Correctness specification given as annotation with precondition and postcondition

$$
\begin{aligned}
&\{\,\varphi\,\}\, P\, \{\,\psi\,\} \\
\iff &\text{“assume } \varphi\text{, execute } P\text{, assert } \psi\text{”} \\
\iff &\varphi \subseteq \mathit{wp}(\psi, P) \\
\iff &\mathit{post}(\varphi, P) \subseteq \psi
\end{aligned}
$$

- Calculus (e.g., *wp*) to automatically derive correctness
  Generates verification conditions

- Strengthen precondition, weaken postcondition

$$
\frac{\{\,\varphi\,\}\, P\, \{\,\psi\,\}}{\{\,\varphi'\,\}\, P\, \{\,\psi'\,\}}\ \varphi' \rightarrow \varphi \text{ and } \psi \rightarrow \psi'
$$

Hoare logic – Loops

- Problematic case: while loop

$$\frac{\{\,\theta \wedge b\,\}\; C_0\; \{\,\theta\,\}}{\{\,\varphi\,\}\; \textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0\; \{\,\psi\,\}}$$

Hoare logic – Loops

- Problematic case: while loop

$$\frac{\{\,\theta \wedge b\,\}\; C_0 \;\{\,\theta\,\}}{\{\,\varphi\,\}\;\textbf{while}\; b \;\textbf{do}\; \{\theta\}\; C_0 \;\{\,\psi\,\}}\; \varphi \rightarrow \theta \text{ and } \theta \wedge \neg b \rightarrow \psi$$

- Remains to show: θ is a loop invariant

Hoare logic – Loops

- Problematic case: while loop

$$\frac{\{\,\theta \wedge b\,\}\; C_0\; \{\,\theta\,\}}{\{\,\varphi\,\}\; \textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0\; \{\,\psi\,\}}\ \varphi \rightarrow \theta \text{ and } \theta \wedge \neg b \rightarrow \psi$$

- Remains to show: θ is a loop invariant

- Annotated loop:
  $wp(\psi, \textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0) =$

Hoare logic – Loops

- Problematic case: while loop

$$\frac{\{\,\theta \wedge b\,\}\; C_0 \;\{\,\theta\,\}}{\{\,\varphi\,\}\; \textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0\; \{\,\psi\,\}}\; \varphi \rightarrow \theta \text{ and } \theta \wedge \neg b \rightarrow \psi$$

- Remains to show: θ is a loop invariant

- Annotated loop:
  "wp"$(\psi, \textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0) = \theta$

- Non-annotated loop:
  $wp(\psi, \textbf{while}\ b\ \textbf{do}\ C_0) =$

Hoare logic – Loops

- Problematic case: while loop

$$\frac{\{\,\theta \wedge b\,\}\; C_0 \,\{\,\theta\,\}}{\{\,\varphi\,\}\,\textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0\,\{\,\psi\,\}}\ \varphi \rightarrow \theta \text{ and } \theta \wedge \neg b \rightarrow \psi$$

- Remains to show: θ is a loop invariant

- Annotated loop:
  $wp(\psi, \textbf{while}\ b\ \textbf{do}\ \{\theta\}\ C_0) = \theta$

- Non-annotated loop:
  $wp(\psi, \textbf{while}\ b\ \textbf{do}\ C_0) = ?$

- Synthesis of loop invariants is a second-order problem

Hoare logic – Relative completeness

- Can we derive any valid partial correctness specification in Hoare calculus automatically?

$$\{\,\varphi\,\}\,P\,\{\,\psi\,\}$$

Hoare logic – Relative completeness

- Can we derive any valid partial correctness specification in Hoare calculus automatically?

$$\{\,\varphi\,\}\, P\, \{\,\psi\,\}$$

- No!

$$\frac{\{\,true\,\}\,\mathit{skip}\,\{\,true\,\}}{\{\,\mathit{true}\,\}\,\textbf{skip}\,\{\,\psi\,\}} \quad true \rightarrow \psi$$

$$\Leftrightarrow \quad \psi \equiv true$$

Hoare logic – Relative completeness

- Can we derive any valid partial correctness specification in Hoare calculus automatically?

$$\{\,\varphi\,\}\,P\,\{\,\psi\,\}$$

- No!

$$\{\,\mathit{true}\,\}\,\textbf{skip}\,\{\,\psi\,\}$$

  Hoare triple valid iff ψ is a tautology

Hoare logic – Relative completeness

- Can we derive any valid partial correctness specification in Hoare calculus automatically?

$$\{\,\varphi\,\}\,P\,\{\,\psi\,\}$$

- No!

$$\{\,\mathit{true}\,\}\,\textbf{skip}\,\{\,\psi\,\}$$

Hoare triple valid iff ψ is a tautology

$$\{\,\mathit{true}\,\}\,P\,\{\,\mathit{false}\,\}$$

Hoare logic – Relative completeness

- Can we derive any valid partial correctness specification in Hoare calculus automatically?

$$\{\,\varphi\,\}\,P\,\{\,\psi\,\}$$

- No!

$$\{\,\mathit{true}\,\}\,\textbf{skip}\,\{\,\psi\,\}$$

  Hoare triple valid iff ψ is a tautology

$$\{\,\mathit{true}\,\}\,P\,\{\,\mathit{false}\,\}$$

  Hoare triple valid iff P does not terminate

Hoare logic – Relative completeness

- Can we derive any valid partial correctness specification in Hoare calculus automatically?

$$\{\,\varphi\,\}\, P\, \{\,\psi\,\}$$

- No!

$$\{\,\mathit{true}\,\}\,\textbf{skip}\,\{\,\psi\,\}$$

  Hoare triple valid iff ψ is a tautology

$$\{\,\mathit{true}\,\}\, P\, \{\,\mathit{false}\,\}$$

  Hoare triple valid iff P does not terminate

- However, we have relative completeness:

$$\models \{\,\varphi\,\}\, P\, \{\,\psi\,\} \implies \vdash \{\,\varphi\,\}\, P\, \{\,\psi\,\}$$

Hoare logic – Soundness & Completeness

- Soundness

$$\vdash \{\,\varphi\,\}\, P \,\{\,\psi\,\} \implies \models \{\,\varphi\,\}\, P \,\{\,\psi\,\}$$

- Completeness

$$\models \{\,\varphi\,\}\, P \,\{\,\psi\,\} \implies \vdash \{\,\varphi\,\}\, P \,\{\,\psi\,\}$$

"Algorithm"?

Hoare logic – Soundness & Completeness

- Soundness

$$\vdash \{\,\varphi\,\}\, P \,\{\,\psi\,\} \implies \models \{\,\varphi\,\}\, P \,\{\,\psi\,\}$$

- Completeness

$$\models \{\,\varphi\,\}\, P \,\{\,\psi\,\} \implies \vdash \{\,\varphi\,\}\, P \,\{\,\psi\,\}$$

"Algorithm":

- Systematically enumerate loop invariant(s) θ
- Annotate P with θ
- Compute $wp(\psi, P)$
- Check $\varphi \subseteq wp(\psi, P)$

Can be interleaved with a search for a counterexample

Overview

Reachable states

- Alternative characterization of safety/correctness

Reachable states

- Alternative characterization of safety/correctness
- No error state is reachable
  i.e., $\varphi_{\mathsf{reach}} \cap \varphi_{\mathsf{err}} = \emptyset$

Reachable states

- Alternative characterization of safety/correctness
- No error state is reachable
  i.e., $\varphi_{\text{reach}} \cap \varphi_{\text{err}} = \emptyset$
- $\varphi_{\text{reach}} = \varphi_{\text{init}} \cup \bigcup_i post^i(\varphi_{\text{init}}, \rho)$
  In general not computable

Reachable states

- Alternative characterization of safety/correctness
- No error state is reachable
  i.e., $\varphi_{\text{reach}} \cap \varphi_{\text{err}} = \emptyset$
- $\varphi_{\text{reach}} = \varphi_{\text{init}} \cup \bigcup_i post^i(\varphi_{\text{init}}, \rho)$
  In general not computable
- Overapproximation: Find a set $\varphi \supseteq \varphi_{\text{reach}}$

Reachable states

- Alternative characterization of safety/correctness
- No error state is reachable
  i.e., $\varphi_{\text{reach}} \cap \varphi_{\text{err}} = \emptyset$
- $\varphi_{\text{reach}} = \varphi_{\text{init}} \cup \bigcup_i post^i(\varphi_{\text{init}}, \rho)$
  In general not computable
- Overapproximation: Find a set $\varphi \supseteq \varphi_{\text{reach}}$
  Nonreachability properties of φ transfer to φ_{reach}

Reachable states

- Alternative characterization of safety/correctness
- No error state is reachable
  i.e., $\varphi_{\text{reach}} \cap \varphi_{\text{err}} = \emptyset$
- $\varphi_{\text{reach}} = \varphi_{\text{init}} \cup \bigcup_i post^i(\varphi_{\text{init}}, \rho)$
  In general not computable
- Overapproximation: Find a set $\varphi \supseteq \varphi_{\text{reach}}$
  Nonreachability properties of φ transfer to φ_{reach}
- Two questions:
  1. How can we find φ?
  2. How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?

Inductive invariants

- How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?

Inductive invariants

- How can we check that $\varphi \supseteq \varphi_{\mathsf{reach}}$ if we do not know φ_{reach}?
- In general, we cannot check this
  But we can check a sufficient condition

Inductive invariants

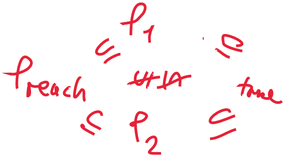
- How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?
- In general, we cannot check this
  But we can check a sufficient condition
- Check that φ is an inductive invariant:
  - $\varphi_{\text{init}} \subseteq \varphi$
  - $post(\varphi, \rho) \subseteq \varphi$

Inductive invariants

- How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?
- In general, we cannot check this
  But we can check a sufficient condition
- Check that φ is an inductive invariant:
  - $\varphi_{\text{init}} \subseteq \varphi$
  - $post(\varphi, \rho) \subseteq \varphi$
- Why is this sufficient?

Inductive invariants

- How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?
- In general, we cannot check this
  But we can check a sufficient condition
- Check that φ is an inductive invariant:
  - $\varphi_{\text{init}} \subseteq \varphi$
  - $post(\varphi, \rho) \subseteq \varphi$



- Why is this sufficient?
  φ_{reach} is the strongest (i.e., smallest) inductive invariant

Inductive invariants

- How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?
- In general, we cannot check this
  But we can check a sufficient condition
- Check that φ is an inductive invariant:
  - $\varphi_{\text{init}} \subseteq \varphi$
  - $post(\varphi, \rho) \subseteq \varphi$
- Why is this sufficient?
  φ_{reach} is the strongest (i.e., smallest) inductive invariant

  What is the weakest inductive invariant?

Inductive invariants

- How can we check that $\varphi \supseteq \varphi_{\text{reach}}$ if we do not know φ_{reach}?
- In general, we cannot check this
  But we can check a sufficient condition
- Check that φ is an inductive invariant:
  - $\varphi_{\text{init}} \subseteq \varphi$
  - $post(\varphi, \rho) \subseteq \varphi$
- Why is this sufficient?
  φ_{reach} is the strongest (i.e., smallest) inductive invariant

  What is the weakest inductive invariant? *true*

Finding inductive invariants

- How can we find an inductive invariant φ?

Finding inductive invariants

- How can we find an inductive invariant φ?
- We want to compute it, so its representation and computation should be finite

Finding inductive invariants

- How can we find an inductive invariant φ?
- We want to compute it, so its representation and computation should be finite
- Abstract interpretation
  We use the instantiation predicate abstraction

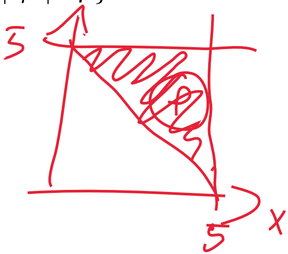
Predicate abstraction

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in \mathit{Preds} \mid \varphi \models p\}$



$p_1: x \leq 5$

$p_2: y \leq 5$

$p_3: x + y \geq 5$

$p_4: x < 4$

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in \mathit{Preds} \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in \mathit{Preds} \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$
- Abstract successor function $\mathit{post}^{\#}(\varphi) := \alpha(\mathit{post}(\varphi))$

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in \mathit{Preds} \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$
- Abstract successor function $\mathit{post}^{\#}(\varphi) := \alpha(\mathit{post}(\varphi))$
- Compute $\varphi^{\#}_{\text{reach}}$: abstract reachability graph

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in \mathit{Preds} \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$
- Abstract successor function $\mathit{post}^{\#}(\varphi) := \alpha(\mathit{post}(\varphi))$
- Compute $\varphi^{\#}_{\text{reach}}$: abstract reachability graph
- Fixpoint reached after finitely many iterations

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in Preds \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$
- Abstract successor function $post^{\#}(\varphi) := \alpha(post(\varphi))$
- Compute $\varphi^{\#}_{\text{reach}}$: abstract reachability graph
- Fixpoint reached after finitely many iterations
- Overapproximation: $\varphi_{\text{reach}} \subseteq \varphi^{\#}_{\text{reach}}$
  $\varphi^{\#}_{\text{reach}}$ is strongest inductive invariant expressible with *Preds*

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in Preds \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$
- Abstract successor function $post^{\#}(\varphi) := \alpha(post(\varphi))$
- Compute $\varphi^{\#}_{\text{reach}}$: abstract reachability graph
- Fixpoint reached after finitely many iterations
- Overapproximation: $\varphi_{\text{reach}} \subseteq \varphi^{\#}_{\text{reach}}$
  $\varphi^{\#}_{\text{reach}}$ is strongest inductive invariant expressible with *Preds*
- "$Preds = \emptyset$"

$\alpha(\varphi_{init})$

Predicate abstraction

- Dynamic building blocks: finite set of predicates *Preds*
- Abstraction function $\alpha : \varphi \mapsto \bigwedge\{p \in Preds \mid \varphi \models p\}$
  - extensive: $\varphi \subseteq \alpha(\varphi)$
  - monotonic: $\varphi \subseteq \psi \implies \alpha(\varphi) \subseteq \alpha(\psi)$
  - idempotent: $\alpha(\varphi) = \alpha(\alpha(\varphi))$
- Abstract successor function $post^{\#}(\varphi) := \alpha(post(\varphi))$
- Compute $\varphi^{\#}_{\text{reach}}$: abstract reachability graph
- Fixpoint reached after finitely many iterations
- Overapproximation: $\varphi_{\text{reach}} \subseteq \varphi^{\#}_{\text{reach}}$
  $\varphi^{\#}_{\text{reach}}$ is strongest inductive invariant expressible with *Preds*
- "$Preds = \emptyset$" is the weakest inductive invariant

(Counterexample-guided) Abstraction refinement

- If abstraction is too coarse, we get spurious counterexamples, i.e., error traces in abstract reachability graph
- Check feasibility of one counterexample
- If infeasible, use it to refine abstraction

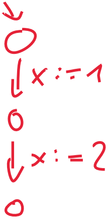
(Counterexample-guided) Abstraction refinement

- If abstraction is too coarse, we get spurious counterexamples, i.e., error traces in abstract reachability graph
- Check feasibility of one counterexample
- If infeasible, use it to refine abstraction
- For example, use *post* or *wp* to compute new predicates
- Recompute abstraction and repeat

Overview

Concept

x := 1
x := 2



Concept

- Consider program as set of traces
- Show that all program traces are infeasible
- Trace τ is infeasible if it satisfies $\{\,\mathit{true}\,\}\,\tau\,\{\,\mathit{false}\,\}$
- Construct finite union of sets of infeasible traces and show containment of all program traces

Automata

- Instantiate concept using finite automata

Ap

$$\mathcal{L}(P) \subseteq \bigcup_i \mathcal{L}(A_i)$$

- Alphabet = set of statements

Automata

- Instantiate concept using finite automata

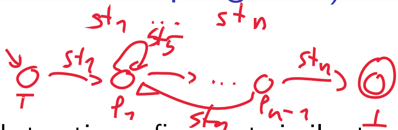
$$\mathcal{L}(P) \subseteq \bigcup_i \mathcal{L}(A_i)$$

- Alphabet = set of statements
- Set of traces of P is in general not regular ($\rightarrow$ abstraction)
- Find counterexample trace in $\mathcal{L}(P) \setminus \bigcup_i \mathcal{L}(A_i)$
- Counterexample can be feasible or infeasible

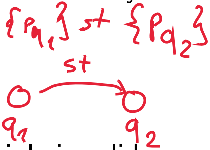
(Counterexample-guided) Abstraction refinement

- Abstraction refinement similar to predicate abstraction?

(Counterexample-guided) Abstraction refinement



- Abstraction refinement similar to predicate abstraction?
- Construct Floyd-Hoare automaton that generalizes infeasibility proof
- Each location is annotated with a predicate
- A transition can be added if the respective Hoare triple is valid



(Counterexample-guided) Abstraction refinement

$A_P \setminus \bigcup_i A_i$

- Abstraction refinement similar to predicate abstraction
- Construct Floyd-Hoare automaton that generalizes infeasibility proof
- Each location is annotated with a predicate
- A transition can be added if the respective Hoare triple is valid
- Output of refinement: automaton, but no predicates

Trace abstraction vs. inductive invariants

- Can we obtain a Hoare annotation of the original program?

Trace abstraction vs. inductive invariants

- Can we obtain a Hoare annotation of the original program?

  Yes: The annotation for a location is the disjunction of the predicates used in the Floyd-Hoare automata

Trace abstraction vs. inductive invariants

- Can we obtain a Hoare annotation of the original program?

  Yes: The annotation for a location is the disjunction of the predicates used in the Floyd-Hoare automata

- This annotation is a safe inductive invariant

Overview

Program verification

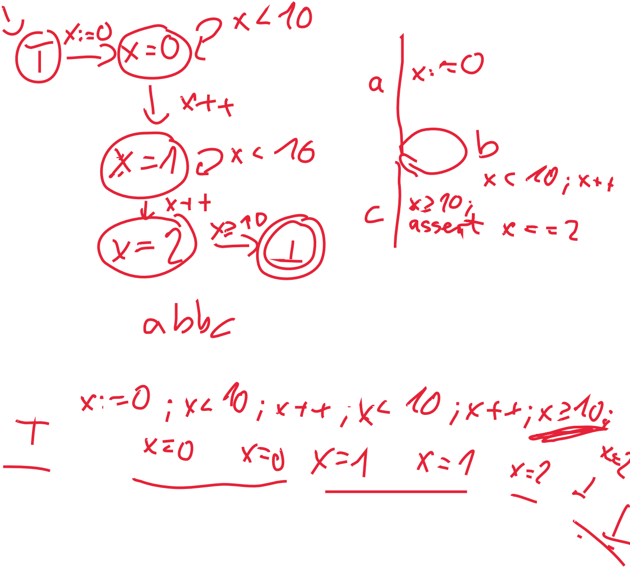
Hoare logic

Abstract reachability

Trace abstraction

Termination

Conclusion



Ranking functions

- A program terminates iff every execution terminates
- A program terminates iff there exists a ~~total~~ ranking function (total)
  - Maps to a well-founded set (= no infinite sequence) $\mathbb{N}$
  - Is strictly decreasing
- We may need to use ordinals (ω)
- Arguments for several variables often use lexicographic ranking functions
- In general, deciding termination is not possible ($\rightarrow$ halting problem)

→ y := 1
x := ?
while (x ≥ 0)
x--

From states to transitions

- Correctness – safe reachable states φ_{reach}
  Termination – well-founded transition relation R_P

From states to transitions

- Correctness – safe reachable states φ_{reach}
  Termination – well-founded transition relation R_P
- We cannot directly show well-foundedness of R_P

From states to transitions

- Correctness – safe reachable states φ_{reach}
  Termination – well-founded transition relation R_P
- We cannot directly show well-foundedness of R_P
- Transition invariant T

$$R_P^+ \subseteq T$$

From states to transitions

- Correctness – safe reachable states φ_{reach}
  Termination – well-founded transition relation R_P
- We cannot directly show well-foundedness of R_P
- Transition invariant T

$$R_P^+ \subseteq T$$

- A transition invariant alone is not sufficient to prove termination

From states to transitions

- Correctness – safe reachable states φ_{reach}
  Termination – well-founded transition relation R_P
- We cannot directly show well-foundedness of R_P
- Transition invariant T

$$R_P^+ \subseteq T$$

$\{(x, x') \mid T\}$

- A transition invariant alone is not sufficient to prove termination
- T must be a finite union of well-founded relations

$$T = T_1 \cup \cdots \cup T_n$$

From states to transitions

- Correctness – safe reachable states φ_{reach}
  Termination – well-founded transition relation R_P
- We cannot directly show well-foundedness of R_P
- Transition invariant T

$$R_P^+ \subseteq T$$

- A transition invariant alone is not sufficient to prove termination
- T must be a finite union of well-founded relations

$$T = T_1 \cup \cdots \cup T_n$$

- Combines several ranking functions

Invariants vs. transition invariants

- Inductive (safety) invariant I

$$\varphi_{\text{init}} \subseteq I \;\wedge\; post(I, \rho) \subseteq I$$

- Transition invariant T

$$R_P \subseteq T \;\wedge\; R_P \circ T \subseteq T$$

- ρ and R_P are basically the same

Computing transition invariants

- Goal: disjunctively well-founded relation T s.t. $R_P^+ \subseteq T$
- Can we compute R_P^+?

Computing transition invariants

- Goal: disjunctively well-founded relation T s.t. $R_P^+ \subseteq T$
- Can we compute R_P^+?
  No, R_P^+ is usually infinite, even if it is well-founded
- As usual, we use abstraction, namely abstract transitions

$$\alpha(\rho) = \bigwedge\{p \in \mathit{Preds} \mid \rho \models p\}$$

Same definition as for abstract states (modulo types)

Algorithm

- Assuming a set of predicates *Preds*, we can use a fixpoint algorithm as for abstract states to compute T
- It remains to show that $T = (T_1 \cup \cdots \cup T_n)$ is disjunctively well-founded

  We have not discussed this in detail[2], but there are efficient algorithms for checking well-foundedness of transition relations obtained from predicate abstraction

[2] See slide 28 from July 19.

Reduction to reachability

- We can reduce the question whether T is a transition invariant for program P to the question whether a modification P' of the program satisfies an invariant I

$$R_P^+ \subseteq T \iff P' \models I$$

- We can analyze the right-hand side as usual
- If we find a feasible counterexample to $P' \models I$, we know that T is not a transition invariant for P
- Abstraction refinement: If the backtranslation of counterexample is terminating, we can add another disjunct T_{n+1} which we can compute from the termination argument

Overview

Correctness

Methods to show correctness

Correctness

Methods to show correctness

- Find loop invariants and prove that $\varphi \subseteq wp(\psi, P)$
- Find a safe inductive invariant
- Show that every trace of the program automaton is infeasible

Correctness

Methods to show correctness

- Find loop invariants and prove that $\varphi \subseteq wp(\psi, P)$
- Find a safe inductive invariant
- Show that every trace of the program automaton is infeasible

Counterexample to correctness

Correctness

Methods to show correctness

- Find loop invariants and prove that $\varphi \subseteq wp(\psi, P)$
- Find a safe inductive invariant
- Show that every trace of the program automaton is infeasible

Counterexample to correctness

- Feasible error trace

Termination

Methods to show termination

Termination

Methods to show termination

- Find ranking function
- Find a disjunctively well-founded inductive transition invariant

Termination

Methods to show termination

- Find ranking function
- Find a disjunctively well-founded inductive transition invariant

Counterexample to termination

Termination

Methods to show termination

- Find ranking function
- Find a disjunctively well-founded inductive transition invariant

Counterexample to termination

- Feasible nonterminating trace

Termination

Methods to show termination

- Find ranking function
- Find a disjunctively well-founded inductive transition invariant

Counterexample to termination

- Feasible nonterminating trace
  - Example: lasso form, i.e., finite stem & finite loop

Termination

Methods to show termination

- Find ranking function
- Find a disjunctively well-founded inductive transition invariant

Counterexample to termination

- Feasible nonterminating trace
  - Example: lasso form, i.e., finite stem & finite loop
    Is this complete?

Termination

Methods to show termination

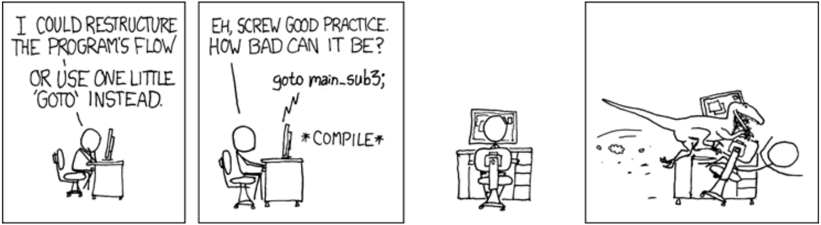
- Find ranking function
- Find a disjunctively well-founded inductive transition invariant

Counterexample to termination

- Feasible nonterminating trace
  - Example: lasso form, i.e., finite stem & finite loop
    Is this complete? No, there are nonterminating programs with only terminating lassos

Have you realized that we used Goto's all the time?

'68 Dijkstra: Go To Statement Considered Harmful
https://doi.org/10.1145%2F362929.362947

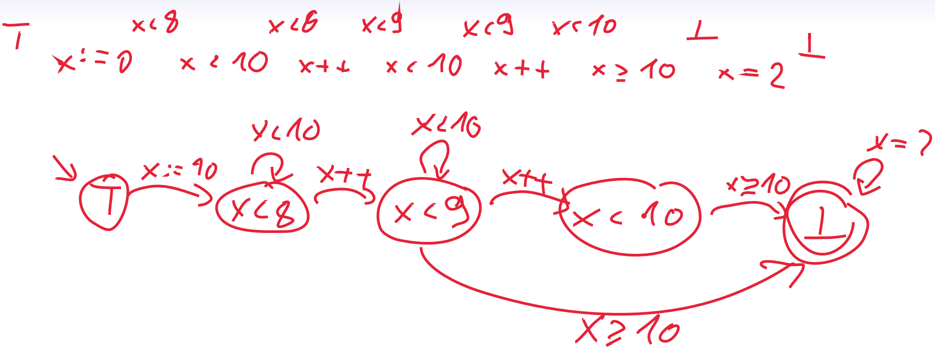


https://xkcd.com/292/

C_1 assume guard ; body

$\{\}$

C_2 assume ¬guard ; rest


⊤
x < 8
x < 8
x < 9
x < 9
x < 10
⊥
⊥
x := 0
x < 10
x++
x < 10
x++
x ≥ 10
x = 2
x < 10
x < 10
x = 2
x := 10
x++
x++
x ≥ 10
⊤
x < 8
x < 9
x < 10
⊥
x ≥ 10