Formal Methods in Software Engineering

An Introduction to Model-Based Analyis and Testing

Vesal Vojdani

Department of Computer Science
University of Tartu

Fall 2014

Orientation

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

What are formal methods?

$$\text{formal method} \quad = \quad \text{formal model} + \text{formal analysis}$$

What is a formal model?

A model is formal if it has. . .

- Well-defined syntax.
- Unambiguous[1] semantics.

[1] mathematical

Formal Analysis

1. Automated Theorem Proving
2. Model Checking
3. Abstract Interpretation

In General

$$\mathcal{M} \models \varphi$$

- $\mathcal{M}$: a situation or model of the system
- φ: a specification of what should hold at situation $\mathcal{M}$

Where do models come from?

1. Hand-written from informal specs.
2. Derived automatically from source code.

Why create a model?

- You can use the model to
  1. analyze if the model behaves well.
  2. test if the implementation conforms to it.
- For this to be worth it, model must be simpler than actual implementation.

Model-Based Analysis

- Model may be simple, but . . .
- execution may be complex (concurrency!)

- Visualize the state graph: manually check functional conformance to informal spec.

- Automatically check all states of the model for safety and liveness properties.

Model-Based Testing

- Automatic test generation requires an oracle.
- The model can be used to automatically generate unit tests with all checks and assertions inserted.
- We can ensure coverage criteria with respect to all states of the model.

Inferring models from code

- The code itself is a formal model!
- It is usually not possible to analyze directly.

- We need bounds and abstractions.

The goal of this course

- Where should you be in one year?
- You are qualified to engage in research to either
  - develop novel verification techniques or
  - apply current techniques in novel contexts.
- Where should you do this work?
  - Our (PLAS) research group!
  - One of the many Estonian companies that are producing novel tools for the maintenance of complex systems.

Must Work Harder

- There will be weekly exercise sheets.
  - They will be made available on Friday.
  - You may ask questions on Wednesday.
  - You will submit electronically on Wednesday evening.
  - We will discuss on Friday.
- Three programming projects.
  - Probably as group work.
  - You may replace this with equivalent thesis work if your supervisor agrees.
- A final exam.

Hoare Logic

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

Hoare Triplets

$$(\!|\, \phi \,|\!)\ P\ (\!|\, \psi \,|\!)$$

- A Hoare triple is satisfied under partial correctness:
  - for each state satisfying ϕ,
  - if execution reaches the end of P,
  - the resulting state satisfies ψ.
- (Total correctness: partial + termination)

Simple Language

$$\begin{aligned} C ::= {} & C_1 \; ; \; C_2 \\ \mid {} & x := e \\ \mid {} & \textsf{if } e \textsf{ then } C_1 \textsf{ else } C_2 \\ \mid {} & \textsf{while } e \textsf{ do } C \\ \mid {} & \textsf{skip} \\ \mid {} & \{C\} \end{aligned}$$

FOL with linear arithmetic

$\phi ::= e$		arithmetic
$\mid$	$\phi_1 \wedge \phi_2$	conjunction
$\mid$	$\phi_1 \vee \phi_2$	disjunction
$\mid$	$\phi_1 \rightarrow \phi_2$	implication
$\mid$	$\exists y : \phi$	existential quantification
$\mid$	$\forall y : \phi$	universal quantification.

Composition

$$\frac{(\!|\ \phi\ |\!)\ C_1\ (\!|\ \eta\ |\!) \qquad (\!|\ \eta\ |\!)\ C_2\ (\!|\ \psi\ |\!)}{(\!|\ \phi\ |\!)\ C_1\ ;\ C_2\ (\!|\ \psi\ |\!)}$$

Assignment

$$\overline{(\!|\, \psi[e/x] \,|\!)\ x = e\ (\!|\, \psi \,|\!)}$$

- Is this backwards?
- Consider examples for $x := 2$ and $x := x + 1$.

Conditional Statements

$$\frac{(\!|\, \phi \wedge e \,|\!)\ C_1\ (\!|\, \psi \,|\!) \qquad (\!|\, \phi \wedge \neg e \,|\!)\ C_2\ (\!|\, \psi \,|\!)}{(\!|\, \phi \,|\!)\ \text{if } e \text{ then } C_1 \text{ else } C_2\ (\!|\, \psi \,|\!)}$$

While Statements

$$\frac{(\!|\ \phi \wedge e\ |\!)\ C\ (\!|\ \phi\ |\!)}{(\!|\ \phi\ |\!)\ \text{while}\ e\ \text{do}\ C\ (\!|\ \phi \wedge \neg e\ |\!)}$$

Implication

$$\frac{\phi' \Rightarrow \phi \qquad (\!|\, \phi \,|\!)\ C\ (\!|\, \psi \,|\!) \qquad \psi \Rightarrow \psi'}{(\!|\, \phi' \,|\!)\ C\ (\!|\, \psi' \,|\!)}$$

- These end up as verification conditions.
- Automated theorem provers have to dispatch them.

Hello World!

```
int abs(int i) {
    if (0 <= i)
        r := i;
    else
        r := -i;
}
```

- Prove: always returns a non-negative value.
- (Where exactly would an overflow invalidate this proof?)

Step by step

1. We first have the conditional:

$$\frac{(\!|\, 0 \leqslant i \,|\!)\ r := i\ (\!|\, 0 \leqslant r \,|\!) \qquad (\!|\, i < 0 \,|\!)\ r := -i\ (\!|\, 0 \leqslant r \,|\!)}{(\!|\, \mathit{true} \,|\!)\ \text{if } 0 \leqslant i \text{ then } r := i \text{ else } r := -i\ (\!|\, 0 \leqslant r \,|\!)}$$

2. The true-branch follows from the assignment axiom.

3. The false-branch relies on a simple implication:

$$\frac{i < 0 \Rightarrow 0 \leqslant -i \qquad (\!|\, 0 \leqslant -i \,|\!)\ r := -i\ (\!|\, 0 \leqslant r \,|\!)}{(\!|\, i < 0 \,|\!)\ r := -i\ (\!|\, 0 \leqslant r \,|\!)}$$

Proof trees

$$\dfrac{(\!| 0 \leqslant i |\!)\ r := i\ (\!| 0 \leqslant r |\!) \qquad \dfrac{i < 0 \Rightarrow 0 \leqslant -i \qquad (\!| 0 \leqslant -i |\!)\ r := -i\ (\!| 0 \leqslant r |\!)}{(\!| i < 0 |\!)\ r := -i\ (\!| 0 \leqslant r |\!)}}{(\!| \mathit{true} |\!)\ \text{if } 0 \leqslant i \text{ then } r := i \text{ else } r := -i\ (\!| 0 \leqslant r |\!)}$$

- The sequential application of inference rules are often represented as proof trees.
- These trees can grow large. . .
- Instead: annotate the program code!
  Tree structure is implicit.

Tableaux Proofs

$$
\begin{array}{l}
\quad (\!|\ \phi_0\ |\!) \\
C_1\ ; \\
\quad (\!|\ \phi_1\ |\!) \\
C_2\ ; \\
\quad (\!|\ \phi_2\ |\!) \\
\quad \vdots \\
\quad (\!|\ \phi_{n-1}\ |\!) \\
C_n \\
\quad (\!|\ \phi_n\ |\!)
\end{array}
$$

Tableaux: Composition

$$\frac{(\!|\, \phi \,|\!)\; C_1 \;(\!|\, \eta \,|\!) \qquad (\!|\, \eta \,|\!)\; C_2 \;(\!|\, \psi \,|\!)}{(\!|\, \phi \,|\!)\; C_1 \,;\, C_2 \;(\!|\, \psi \,|\!)}$$

$$\begin{array}{l} \qquad (\!|\, \phi \,|\!) \\ C_1 \;; \\ \qquad (\!|\, \eta \,|\!) \\ \quad C_2 \\ \qquad (\!|\, \psi \,|\!) \end{array}$$

Tableaux: Conditional

$$\frac{\begin{array}{c}(\!|\, \phi \wedge e \,|\!)\ C_1\ (\!|\, \psi \,|\!) \\ (\!|\, \phi \wedge \neg e \,|\!)\ C_2\ (\!|\, \psi \,|\!)\end{array}}{(\!|\, \phi \,|\!)\ \text{if } e \text{ then } C_1 \text{ else } C_2\ (\!|\, \psi \,|\!)}$$

$$\begin{array}{l}
\quad (\!|\, \phi \,|\!) \\
\text{if } e \text{ then } \{ \\
\qquad\quad (\!|\, \phi \wedge e \,|\!) \\
\quad C_1 \\
\qquad\quad (\!|\, \psi \,|\!) \\
\} \text{ else } \{ \\
\qquad\quad (\!|\, \phi \wedge \neg e \,|\!) \\
\quad C_2 \\
\qquad\quad (\!|\, \psi \,|\!) \\
\} \\
\quad (\!|\, \psi \,|\!)
\end{array}$$

Tableaux: Conditional

$$\frac{(\!|\,\phi \wedge e\,|\!)\; C_1 \;(\!|\,\psi\,|\!) \qquad (\!|\,\phi \wedge \neg e\,|\!)\; C_2 \;(\!|\,\psi\,|\!)}{(\!|\,\phi\,|\!)\ \text{if } e \text{ then } C_1 \text{ else } C_2\ (\!|\,\psi\,|\!)}$$

$(\!|\,\phi\,|\!)$

if e then {

$(\!|\,\phi \wedge e\,|\!)$

C_1

} else {

$(\!|\,\phi \wedge \neg e\,|\!)$

C_2

}

$(\!|\,\psi\,|\!)$

Tableaux: Implication

$$\frac{\phi' \Rightarrow \phi \qquad (\!|\, \phi \,|\!)\ C\ (\!|\, \psi \,|\!) \qquad \psi \Rightarrow \psi'}{(\!|\, \phi' \,|\!)\ C\ (\!|\, \psi' \,|\!)}$$

$$\begin{array}{c} (\!|\, \phi' \,|\!) \\ (\!|\, \phi \,|\!) \\ C \\ (\!|\, \psi \,|\!) \\ (\!|\, \psi' \,|\!) \end{array}$$

The example as tableaux proof

$$
\begin{array}{l}
\quad (\!|\, \mathit{true} \,|\!) \\
\text{if } (0 \leqslant i) \text{ then } \{ \\
\qquad\qquad (\!|\, \mathit{true} \wedge 0 \leqslant i \,|\!) \\
\qquad r := i \\
\qquad\qquad (\!|\, 0 \leqslant r \,|\!) \\
\} \text{ else } \{ \\
\qquad\qquad (\!|\, \mathit{true} \wedge i < 0 \,|\!) \\
\qquad\qquad (\!|\, 0 \leqslant -i \,|\!) \\
\qquad r := -i \\
\qquad\qquad (\!|\, 0 \leqslant r \,|\!) \\
\} \\
\quad (\!|\, 0 \leqslant r \,|\!)
\end{array}
$$

Weakest Pre-Conditions

- We have so far only rules for valid Hoare triples.
- Not all triples are equally useful

$$\llparenthesis\, \mathit{false} \,\rrparenthesis\ \mathrm{P}\ \llparenthesis\, \psi \,\rrparenthesis$$

- How do we infer these triples?
- We will now move towards a more syntax-driven method to infer weakest pre-conditions.

Definition

- We say ϕ is weaker than ϕ' if

$$\phi' \Rightarrow \phi$$

- For $\phi = \mathrm{WP}\,[\![S]\!]\,\psi$, we have

$$(\!|\,\phi\,|\!)\ S\ (\!|\,\psi\,|\!) \text{ is valid}$$
$$\text{if } (\!|\,\phi'\,|\!)\ S\ (\!|\,\psi\,|\!) \text{ then } \phi' \Rightarrow \phi$$

- ψ holds after S iff ϕ holds before.

Assignment

- Consider sequential composition:

$$
\begin{aligned}
z &:= x;\\
z &:= z + y;\\
u &:= z
\end{aligned}
$$

- It suffices with definitions:

$$
\begin{aligned}
\text{WP}\ [\![x = e]\!]\ \psi &= \psi[e/x]\\
\text{WP}\ [\![C_1\ ;\ C_2]\!]\ \psi &= \text{WP}\ [\![C_1]\!]\ (\text{WP}\ [\![C_2]\!]\ \psi)
\end{aligned}
$$

A tableaux proof from WPs

$$
\begin{array}{l}
\quad (\!|\, x + y = 42 \,|\!) \\
z := x; \\
\quad (\!|\, z + y = 42 \,|\!) \\
z := z + y; \\
\quad (\!|\, z = 42 \,|\!) \\
u := z \\
\quad (\!|\, u = 42 \,|\!)
\end{array}
$$

Conditional

- Hoare logic:

$$\frac{(\!|\ \phi \wedge e\ |\!)\ C_1\ (\!|\ \psi\ |\!) \qquad (\!|\ \phi \wedge \neg e\ |\!)\ C_2\ (\!|\ \psi\ |\!)}{(\!|\ \phi\ |\!)\ \text{if } e \text{ then } C_1 \text{ else } C_2\ (\!|\ \psi\ |\!)}$$

- A more syntax-driven rule:

$$\frac{(\!|\ \phi_1\ |\!)\ C_1\ (\!|\ \psi\ |\!) \qquad (\!|\ \phi_2\ |\!)\ C_2\ (\!|\ \psi\ |\!)}{(\!|\ \phi'\ |\!)\ \text{if } e \text{ then } C_1 \text{ else } C_2\ (\!|\ \psi\ |\!)}$$

where $\phi' = (e \rightarrow \phi_1) \wedge (\neg e \rightarrow \phi_2)$

Proof Tableaux for Conditional 2.0

if e then {

C_1

} else {

C_2

}

$(\!| \psi |\!)$

Proof Tableaux for Conditional 2.0

$$
\begin{array}{l}
\text{if } e \text{ then } \{ \\
\qquad\qquad (\!|\ \mathsf{WP}\ [\![C_1]\!]\ \psi\ |\!) \\
\qquad C_1 \\
\}\ \text{else}\ \{ \\
\qquad\qquad (\!|\ \mathsf{WP}\ [\![C_2]\!]\ \psi\ |\!) \\
\qquad C_2 \\
\} \\
\qquad (\!|\ \psi\ |\!)
\end{array}
$$

Proof Tableaux for Conditional 2.0

$$(\!|\ (e \to \mathsf{WP}\,[\![C_1]\!]\,\psi) \wedge (\neg e \to \mathsf{WP}\,[\![C_2]\!]\,\psi)\ |\!)$$

if e then {

$$(\!|\ \mathsf{WP}\,[\![C_1]\!]\,\psi\ |\!)$$

C_1

} else {

$$(\!|\ \mathsf{WP}\,[\![C_2]\!]\,\psi\ |\!)$$

C_2

}

$$(\!|\ \psi\ |\!)$$

The Example Again

$$\text{if } (0 \leqslant i) \text{ then } \{$$

$$r := i$$

$$\} \text{ else } \{$$

$$r := -i$$

$$\}$$

$$(\!| 0 \leqslant r |\!)$$

The Example Again

$$
\begin{array}{l}
\text{if } (0 \leqslant i) \text{ then } \{ \\
\qquad\qquad (\!| 0 \leqslant i |\!) \\
\qquad r := i \\
\} \text{ else } \{ \\
\qquad\qquad (\!| 0 \leqslant -i |\!) \\
\qquad r := -i \\
\} \\
\qquad (\!| 0 \leqslant r |\!)
\end{array}
$$

The Example Again

$$(\!|\, (0 \leqslant i \to 0 \leqslant i) \land (i < 0 \to 0 \leqslant -i) \,|\!)$$

$$\text{if } (0 \leqslant i) \text{ then } \{$$

$$(\!|\, 0 \leqslant i \,|\!)$$

$$r := i$$

$$\} \text{ else } \{$$

$$(\!|\, 0 \leqslant -i \,|\!)$$

$$r := -i$$

$$\}$$

$$(\!|\, 0 \leqslant r \,|\!)$$

The Example Again

$$(\!|\, true \,|\!)$$

$$(\!|\, (0 \leqslant i \to 0 \leqslant i) \land (i < 0 \to 0 \leqslant -i) \,|\!)$$

if $(0 \leqslant i)$ then {

$$(\!|\, 0 \leqslant i \,|\!)$$

$$r := i$$

} else {

$$(\!|\, 0 \leqslant -i \,|\!)$$

$$r := -i$$

}

$$(\!|\, 0 \leqslant r \,|\!)$$

Loop Invariants

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

Warm-Up

- Consider a simple loop-free program:

```
int succ(int x) {
    a = x + 1;
    if (a - 1 == 0)
        y = 1;
    else
        y = a;
    return y;
}
```

- Show that $y = x + 1$ at the return statement.

While Loops

- Recall the proof rule

$$\frac{(\!|\ \phi \wedge e\ |\!)\ C\ (\!|\ \phi\ |\!)}{(\!|\ \phi\ |\!)\ \text{while } e \text{ do } C\ (\!|\ \phi \wedge \neg e\ |\!)}$$

- Given a ψ as post-condition. . .
- How can we apply this rule?
- What is the WP of a while loop?

Termination?

- Weakest Liberal Preconditions

$$wp\,[\![S]\!]\,\psi \equiv wp\,[\![S]\!]\,true \wedge wlp\,[\![S]\!]\,\psi$$

- We did not care about this distinction
  - Termination is an outdated concept. ;)
  - Only loops have different definitions.

WP for while loops

- WP $[\![\text{while } e \text{ do } C]\!]\ \psi$?
- Unrolling the loop:

$$
\begin{aligned}
F_0 &= \text{while } e \text{ do skip} \\
F_i &= \text{if } e \text{ then } C \,;\, F_{i-1} \text{ else skip}
\end{aligned}
$$

- WP for "exiting the loop after at most i iterations in a state satisfying ψ":

$$
\begin{aligned}
L_0 &\equiv \psi \wedge \neg e \\
L_i &\equiv (\neg e \rightarrow \phi) \wedge (e \rightarrow \text{WP } [\![C]\!]\ L_{i-1})
\end{aligned}
$$

WLP for while loops

- WLP $[\![\text{while } e \text{ do } C]\!]\ \psi$?
- Unrolling the loop:

$$F_0 = \text{while } e \text{ do skip}$$
$$F_i = \text{if } e \text{ then } C \text{ ; } F_{i-1} \text{ else skip}$$

- WLP for "if we exit the loop after at most i iterations, the resulting state satisfies ψ":

$$L_0 \equiv \psi$$
$$L_i \equiv (\neg e \rightarrow \phi) \wedge (e \rightarrow \text{WLP } [\![C]\!]\ L_{i-1})$$

WLP for while loops

- WLP for "if we exit the loop after at most i iterations, the resulting state satisfies ψ":

$$L_0 \equiv \psi$$
$$L_i \equiv (\neg e \rightarrow \phi) \wedge (e \rightarrow \text{WLP}\,[\![C]\!]\,L_{i-1})$$

- We then define

$$\text{WLP}\,[\![\text{while } e \text{ do } C]\!]\,\psi = \forall i \in \mathbb{N} : L_i$$

- Not very practical…

Precondition of a While Loop

To push ψ up through $\text{while } e \text{ do } C$:

1. Guess a potential invariant ϕ.
2. Make sure $\phi \land \neg e \implies \psi$.
3. Compute $\phi' = \text{WLP } [\![C]\!] \ \phi$.
4. Check that $\phi \land e \implies \phi'$.
5. Then, ϕ is a pre-condition for ψ.

$$\frac{(\!|\ \phi \land e\ |\!)\ C\ (\!|\ \phi\ |\!)}{(\!|\ \phi\ |\!)\ \text{while } e \text{ do } C\ (\!|\ \phi \land \neg e\ |\!)}$$

Proof Tableaux for Loops

$$
\begin{array}{l}
\qquad (\!|\ \phi\ |\!) \\
\text{while } e \text{ do } \{ \\
\qquad\qquad (\!|\ \phi \wedge e\ |\!) \\
\qquad\qquad (\!|\ \mathrm{WLP}\ [\![C]\!]\ \phi\ |\!) \\
\qquad C \\
\qquad\qquad (\!|\ \phi\ |\!) \\
\} \\
\qquad (\!|\ \phi \wedge \neg E\ |\!) \\
\qquad (\!|\ \psi\ |\!)
\end{array}
$$

Exercise 1

```
int fact(int x) {
    y = 1;
    z = 0;
    while (z != x) {
        z = z + 1;
        y = y * z;
    }
    return y;
}
```

Guessing the invariant

- Doing a trace:

iteration	x	y	z	B
0	6	1	0	$true$
1	6	1	1	$true$
2	6	2	2	$true$
3	6	6	3	$true$
4	6	24	4	$true$
5	6	120	5	$true$
6	6	720	6	$false$
i		$i!$	i	

- Formulate hypothesis: $y = z!$

Proof obligations

Want to establish $\psi \equiv y = x!$.

1. Our invariant $\phi \equiv y = z!$
2. Check that $\phi \wedge \neg(z \neq x) \implies \psi$.

Proof obligations

Want to establish $\psi \equiv y = x!$.

1. Our invariant $\phi \equiv y = z!$
2. Check that $\phi \wedge \neg(z \neq x) \implies \psi$.
3. Compute WLP of loop body:

Proof obligations

Want to establish $\psi \equiv y = x!$.

1. Our invariant $\phi \equiv y = z!$
2. Check that $\phi \wedge \neg(z \neq x) \implies \psi$.
3. Compute WLP of loop body:

$$\phi' \equiv y \cdot (z+1) = (z+1)!$$

4. Check if $\phi \wedge z \neq x \implies \phi'$.

Proof obligations

Want to establish $\psi \equiv y = x!$.

1. Our invariant $\phi \equiv y = z!$
2. Check that $\phi \wedge \neg(z \neq x) \implies \psi$.
3. Compute WLP of loop body:

$$\phi' \equiv y \cdot (z+1) = (z+1)!$$

4. Check if $\phi \wedge z \neq x \implies \phi'$.
5. Continue WLP computation with ϕ.

Exercise 2: Minimal-Sum Section

- Given an integer array $a[0], a[1], \ldots, a[n-1]$.
- A section of a is a continuous piece $a[i], a[i+1], \ldots, a[j]$ with $0 \leqslant i \leqslant j < n$.
- Section sum: $S_{i,j} = a[i] + \cdots + a[j]$.
- A minimal-sum section is a section $a[i], \ldots, a[j]$ s.t. for any other $a[i'], \ldots, a[j']$, we have $S_{i,j} \leqslant S_{i',j'}$.

What to do?

- Compute the sum of the minimal-sum sections in linear time.
- Prove that the code is correct!
- For example...
  - $[-1, 3, 15, -6, 4, -5]$ is -7 for $[-6, 4, -5]$.
  - $[-2, -1, 3, -3]$ is -3 for $[-2, -1]$ or $[-3]$.

The Program

```
int minsum(int a[]) {
    k = 1;
    t = a[0];
    s = a[0];
    while (k != n) {
        t = min(t + a[k], a[k]);
        s = min(s,t);
        k = k + 1;
    }
    return s;
}
```

Post-conditions

- The value s is smaller than the sum of any section.

$$\phi_1 = \forall i, j : 0 \leqslant i \leqslant j < n \rightarrow s \leqslant S_{i,j}$$

- There is a section whose sum is s

$$\phi_2 = \exists i, j : 0 \leqslant i \leqslant j < n \wedge s = S_{i,j}$$

Trying to prove ϕ_1

- Suitable Invariant:

$$\phi_1 = \forall i, j : 0 \leqslant i \leqslant j < n \rightarrow s \leqslant S_{i,j}$$
$$I_1(s, k) = \forall i, j : 0 \leqslant i \leqslant j < k \rightarrow s \leqslant S_{i,j}$$

Trying to prove ϕ_1

- Suitable Invariant:

$$\phi_1 = \forall i, j : 0 \leqslant i \leqslant j < n \rightarrow s \leqslant S_{i,j}$$
$$I_1(s, k) = \forall i, j : 0 \leqslant i \leqslant j < k \rightarrow s \leqslant S_{i,j}$$

- Additional Invariant

$$I_2(t, k) = \forall i : 0 \leqslant i < k \rightarrow t \leqslant S_{i,k-1}$$

The Key Lemma

- In the end, we have to prove that

$$I_1(s, k) \wedge I_2(t, k) \wedge k \neq n$$
$$\Longrightarrow$$
$$I_1(\min(s, (\min(t + a[k], a[k])), k + 1)$$
$$\wedge$$
$$I_2(\min(t + a[k], a[k]), k + 1)$$

- This will require human intervention: proof-assistants.

Verification Condition Generation

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

Purpose of this lecture

- Get an idea of how verification condition generation works.

- We consider the simplest possible implementation.
- This is based on early work on ESC/Java.

- We see some important concepts:
  - collecting semantics
  - constraint systems
  - abstraction

Quick: What is the Loop Invariant?

$$
\begin{array}{l}
y := 5 \; ; \\
x := 0 \; ; \\
\textsf{while}\ x \neq 5\ \textsf{do} \\
\qquad x := x + 1 \\
\quad (\!| \, x = y \, |\!)
\end{array}
$$

Generating VCs

- Non-trivial loop-invariants must be supplied, but everything else automatic.
- Assume program is annotated with
  - Pre- & Post-conditions.
  - For every while-loop, a supposed loop-invariant.
- How do we check automatically that the implementation satisfies the contract?

Verification Conditions

- Consider the triplets:

$$(\!|\, \phi \,|\!)\ C\ (\!|\, \psi \,|\!)$$
$$(\!|\, x = x' \,|\!)\ x := x - y\ (\!|\, x + y = x' \,|\!)$$

- The verification conditions would be

$$\phi \rightarrow \mathrm{WP}\ [\![C]\!]\ \psi$$
$$(x = x') \rightarrow ((x - y) + y = x')$$

Asking an SMT Solver

- We then ask an SMT solver if the VC is true.

$$(x = x') \rightarrow ((x - y) + y = x')$$

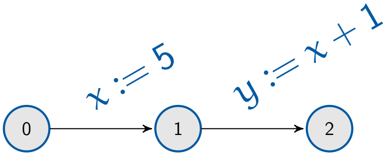
- We want the VC to hold for all parameters.
- Check if the negated formula is satisfiable!

- Think: searching for a falsifying assignment (failing test case).

Translation into Flow Graphs

Control Flow Graph $G = (N, E, s, r)$

- N are program points, and $s, r \in N$ are start/return nodes.
- $E = N \times C \times N$ are transition, where C is the set of basic statements.



Basic Edges

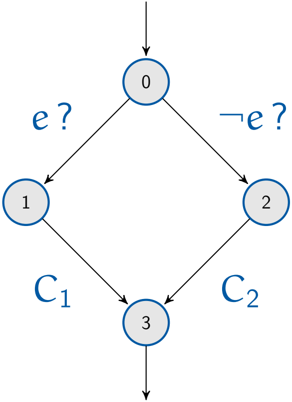
$C ::=$	skip	skip
$\mid$	$x := e$	assign
$\mid$	$\phi\,?$	assume
$\mid$	$\phi\,!$	assert

FOL with linear arithmetic

$\phi ::= e$		arithmetic
$\mid$	$\phi_1 \wedge \phi_2$	conjunction
$\mid$	$\phi_1 \vee \phi_2$	disjunction
$\mid$	$\phi_1 \rightarrow \phi_2$	implication
$\mid$	$\exists y : \phi$	existential quantification
$\mid$	$\forall y : \phi$	universal quantification.

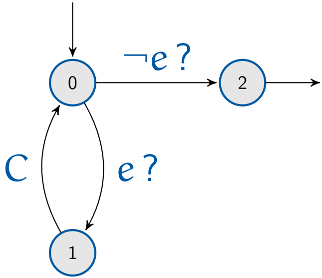
Translating If-Statements

$$\text{if } e \text{ then } C_1 \text{ else } C_2$$



Translating While-Statements

while e do C



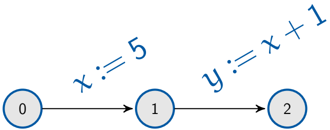
Program State

- State σ assigns values to variables:

$$\sigma\colon V \to \mathbb{Z}$$

- Example:

$$\sigma_0 = \{x \mapsto 0, y \mapsto 0\}$$



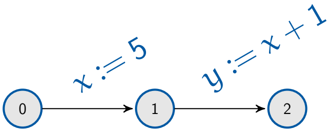
Program State

- State σ assigns values to variables:

$$\sigma \colon V \to \mathbb{Z}$$

- Example:

$$\sigma_0 = \{x \mapsto 0, y \mapsto 0\}$$
$$\sigma_1 = \{x \mapsto 5, y \mapsto 0\}$$



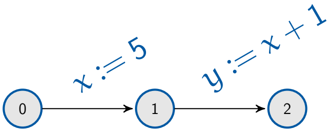
Program State

- State σ assigns values to variables:

$$\sigma\colon V \to \mathbb{Z}$$

- Example:

$$\sigma_0 = \{x \mapsto 0, y \mapsto 0\}$$
$$\sigma_1 = \{x \mapsto 5, y \mapsto 0\}$$
$$\sigma_2 = \{x \mapsto 5, y \mapsto 6\}$$



Evaluating Expressions

- Given a σ, we evaluate expressions:

$$
\begin{aligned}
[\![z]\!]\,\sigma &= z \\
[\![x]\!]\,\sigma &= \sigma\,x \\
[\![e_1 + e_2]\!]\,\sigma &= [\![e_1]\!]\,\sigma + [\![e_2]\!]\,\sigma \\
&\dots
\end{aligned}
$$

- For $\sigma = \{x \mapsto 5, y \mapsto 6\}$,

$$
[\![x + y]\!]\,\sigma = [\![x]\!]\,\sigma + [\![y]\!]\,\sigma = \\
\sigma\,x + \sigma\,y = 5 + 6 = 11
$$

State satisfies a formula

- Our state is $\sigma\colon V \to \mathbb{Z}$, but ϕ may contain unbound logical variables $x' \notin V$.
- A state σ satisfies ϕ

$$\sigma \vDash \phi$$

  if ϕ evaluates to true for some extension of σ:

$$\exists \sigma' : (\forall v \in V : \sigma' v = \sigma v) \wedge (\llbracket \phi \rrbracket \sigma' = \mathit{true})$$

- And a formula ϕ is satisfiable if $\exists \sigma : \sigma \vDash \phi$.

A note on triplets

- Consider the triplet

$$(\!|\, x = x' \,|\!)\ x := x + 1\ (\!|\, x = x' + 1 \,|\!)$$

  where x' is a logical variable.

- When we say that the triplet $(\!|\, \phi \,|\!)\ C\ (\!|\, \psi \,|\!)$ is valid under partial correctness if

$$\forall \sigma : \sigma \vDash \phi \implies [\![C]\!]\sigma \vDash \psi$$

  we assume that σ includes logical variables.

Notation: Updating the State

- We update the mapping σ:

$$\sigma' = \sigma[x \mapsto z]$$

where

$$\sigma'\,y = \begin{cases} z & \text{if } y = x \\ \sigma\,y & \text{otherwise} \end{cases}$$

- Useful exercise:

$$\sigma \models \psi[e/x] \iff \sigma[x \mapsto [\![e]\!]\,\sigma] \models \psi$$

Notation: Updating the State

- We update the mapping σ:

$$\sigma' = \sigma[x \mapsto z]$$

  where

$$\sigma'\, y = \begin{cases} z & \text{if } y = x \\ \sigma\, y & \text{otherwise} \end{cases}$$

- Useful exercise:

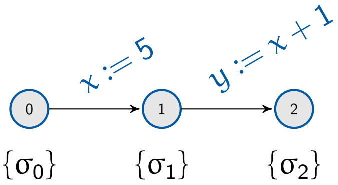
$$\sigma \vDash \psi[e/x] \iff \sigma[x \mapsto [\![e]\!]\,\sigma] \vDash \psi$$

$$[\![\psi[e/x]]\!]\,\sigma = [\![\psi]\!]\,(\sigma[x \mapsto [\![e]\!]\,\sigma])$$

Collecting Semantics

- For every point $p \in N$, we want to know
- The set of states reaching p: S_p.
- If we assume that $S_s = S_0 = \{\sigma_0\}$.

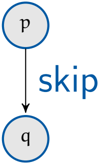
$$\sigma_0\ v = 0 \quad (\forall v \in V)$$



Starting State

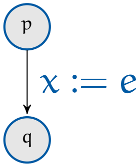
- We need this semantics to validate our WP computation.
- Therefore, the best choice is $S_s = V \to \mathbb{Z}$, so that only tautologies hold at s.
- We include all logical variables from assume statements in V.

For a skip edge



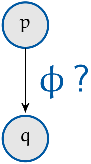
$$S_q = S_p$$

For an assignment edge



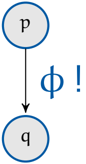
$$S_q = \{\sigma[x \mapsto [\![e]\!]\sigma] \mid \sigma \in S_p\}$$

For an assume edge



$$S_q = \{\sigma \mid \sigma \in S_p,\ [\![\phi]\!]\,\sigma = \mathit{true}\}$$

For an assert edge



$$S_q = \{\sigma \mid \sigma \in S_p,\ [\![\phi]\!]\,\sigma = \mathit{true}\} \\ \cup \{\bot \mid \sigma \in S_p,\ [\![\phi]\!]\,\sigma = \mathit{false}\}$$

Quiz: The Error State

- For any S, what are the results of the edges?

$$\mathit{false}\,? \qquad \mathit{false}\,!$$

Quiz: The Error State

- For any S, what are the results of the edges?

$\mathit{false}\,?$	$\mathit{false}\,!$
$\emptyset$	$\{\bot\}$

Quiz: The Error State

- For any S, what are the results of the edges?

$$\begin{array}{cc} \mathit{false}\,? & \mathit{false}\,! \\ \emptyset & \{\bot\} \end{array}$$

- The "$\bot$" should pass through other edges (like exceptions / maybe monad)

$$[\![\phi]\!]\,\bot = \mathit{false} \qquad \bot\,[x \mapsto e] = \bot$$

- We amend the assume rule…

Transfer functions

$$\llbracket \mathsf{skip} \rrbracket\, S = S$$

$$\llbracket x := e \rrbracket\, S = \{\sigma\,[x \mapsto \llbracket e \rrbracket\,\sigma] \mid \sigma \in S\}$$

$$\begin{aligned} \llbracket e\,? \rrbracket\, S = \{&\sigma \mid \sigma \in S_p,\ \llbracket e \rrbracket\,\sigma \neq 0\} \\ \cup\ \{&\bot \mid \bot \in S_p\} \end{aligned}$$

$$\begin{aligned} \llbracket e\,! \rrbracket\, S = \{&\sigma \mid \sigma \in S_p,\ \llbracket e \rrbracket\,\sigma \neq 0\} \\ \cup\ \{&\bot \mid \sigma \in S_p,\ \llbracket e \rrbracket\,\sigma = 0\} \end{aligned}$$

Satisfiability for Sets

- This is lifted as expected:

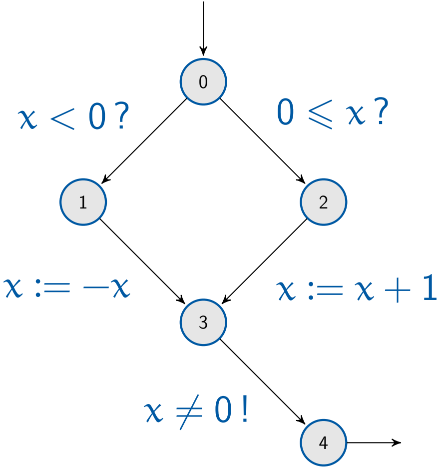
$$S \models \phi \iff \forall \sigma \in S : \sigma \models \phi$$

- As the error state satisfies nothing:

$$\forall \phi : \bot \not\models \phi$$

- if $\bot \in S$, already $S \not\models \mathit{true}$.
  (because some assertions may already have failed.)

Example



Equation & Constraint Systems

- Recall $G = (N, E, s, r)$.
- First we set the starting state:

$$S_s = \{\sigma_s\} \qquad \text{(or } S_s = V \to \mathbb{Z}\text{)}$$

And for each point $q \in N$:

$$S_q = \bigcup \{ [\![C]\!]\, S_p \mid (p, C, q) \in E \}$$

Equation & Constraint Systems

- Recall $G = (N, E, s, r)$.
- First we set the starting state:

$$S_s = \{\sigma_s\} \qquad \text{(or } S_s = V \to \mathbb{Z}\text{)}$$

And for each point $q \in N$:

$$S_q = \bigcup \{[\![C]\!]\, S_p \mid (p, C, q) \in E\}$$

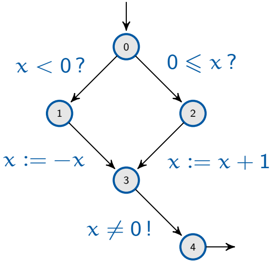
- As a constraint system:

$$S_s \supseteq \{\sigma_s\}$$
$$S_q \supseteq [\![C]\!]\, S_p \qquad \text{for } (p, C, q) \in E$$

Constraint System Example

- Let $x_p = \{\sigma\, x \mid \sigma \in S_p\}$ (and $\perp$ if $\sigma = \perp$).
- We start with $x_0 = x_s = \mathbb{Z}$.

$$
\begin{aligned}
x_0 &\supseteq \mathbb{Z} \\
x_1 &\supseteq \{z \mid z \in x_0,\ z < 0\} \\
x_2 &\supseteq \{z \mid z \in x_0,\ 0 \leqslant z\} \\
x_3 &\supseteq \{-z \mid z \in x_1\} \\
x_3 &\supseteq \{z + 1 \mid z \in x_2\} \\
x_4 &\supseteq \{z \mid z \in x_3,\ z \neq 0\} \\
&\quad \cup \{\perp \mid z \in x_3,\ z = 0\}
\end{aligned}
$$



And Now WP…

$$
\begin{aligned}
\mathrm{WP}\,[\![\mathsf{skip}]\!]\;\psi &= \psi \\
\mathrm{WP}\,[\![x := e]\!]\;\psi &= \psi[e/x] \\
\mathrm{WP}\,[\![\phi\,?]\!]\;\psi &= \phi \to \psi \\
\mathrm{WP}\,[\![\phi\,!]\!]\;\psi &= \phi \land \psi
\end{aligned}
$$

Assume versus Assert

- Definitions:

C	$wp[\![C]\!]\,\psi$	$wlp[\![C]\!]\,\psi$
$\phi\,!$	$\phi \wedge \psi$	$\phi \rightarrow \psi$
$\phi\,?$	$\phi \rightarrow \psi$	$\phi \rightarrow \psi$

- Our WP $[\![C]\!]\,\psi$ behaves like wp on asserts.
- However, we will abstract away loops, so in essence this is still partial correctness.

Equation system for WP

- We now start from the end node $r \in N$.
- Post-conditions are explicitly asserted, so...
- We start with $\psi_r = \mathit{true}$ and for $p \in N$:

$$\psi_p = \bigwedge \{ \mathrm{WP}\,[\![c]\!]\,\psi_q \mid (p, c, q) \in E \}$$

- Alternatively, as a constraint system:

$$\begin{aligned} \psi_r &\implies \mathit{true} \\ \psi_p &\implies \mathrm{WP}\,[\![c]\!]\,\psi_q \qquad \text{for } (p, c, q) \in E \end{aligned}$$

WP and our Semantics

- Assume we have computed the initial precondition ψ_s starting from the end node $\psi_r = true$.
- If we start the collecting semantics with

$$S_s = \{\sigma \mid \sigma \models \psi_s\}$$

- Then, we expect:

$$S_r \models true$$

which holds whenever $\bot \notin S_r$.

Quiz: Error State Again

- Recall our false assume/assert edges:

$$\begin{array}{cc} \mathit{false}\,? & \mathit{false}\,! \\ \emptyset & \{\bot\} \end{array}$$

- Now what is the WP for these?

$$\mathrm{WP}\,[\![\mathit{false}\,?]\!]\,\psi \qquad \mathrm{WP}\,[\![\mathit{false}\,!]\!]\,\psi$$

Quiz: Error State Again

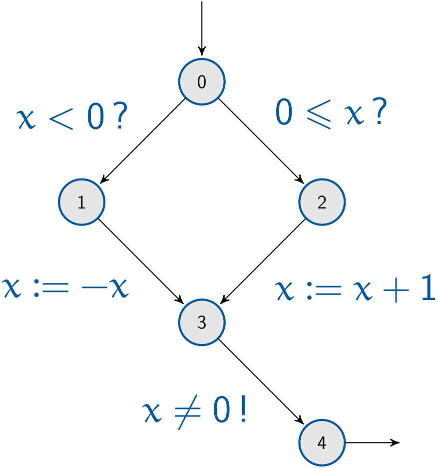
- Recall our false assume/assert edges:

$$\begin{array}{cc} \mathit{false}\,? & \mathit{false}\,! \\ \emptyset & \{\bot\} \end{array}$$

- Now what is the WP for these?

$$\begin{array}{cc} \text{WP}\,[\![\mathit{false}\,?]\!]\;\psi & \text{WP}\,[\![\mathit{false}\,!]\!]\;\psi \\ \mathit{true} & \mathit{false} \end{array}$$

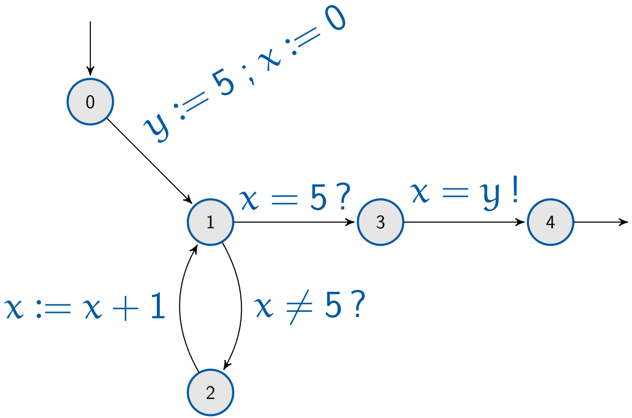
Again this example:



Now recall this example. . .

$$
\begin{aligned}
&y := 5\ ;\\
&x := 0\ ;\\
&\text{while}\ x \neq 5\ \text{do}\\
&\qquad x := x + 1\ ;\\
&x = y\ !
\end{aligned}
$$

We could compute this. . .



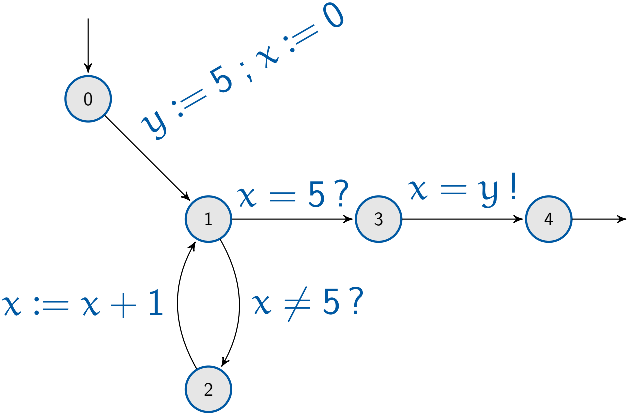
VCG: Abstraction of Loops

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

WP computation was stuck in this loop



Havoc (wrong!)

- Concrete semantics:

$$[\![\mathsf{havoc}\, x]\!]\, S = \{\sigma\,[x \mapsto z] \mid \sigma \in S,\ z \in \mathbb{Z}\}$$

- WP for havoc:

$$\mathsf{WP}\ [\![\mathsf{havoc}\, x]\!]\ \psi = \exists x : \psi$$

- Practically, all information about x is lost, except indirect relations remain:

$$\mathsf{WP}\ [\![\mathsf{havoc}\, x]\!]\ (y = x \land x = z) \implies (y = z)$$

Havoc (for post-conditions!)

- Concrete semantics:

$$[\![\mathsf{havoc}\, x]\!]\, S = \{\sigma[x \mapsto z] \mid \sigma \in S,\ z \in \mathbb{Z}\}$$

- WP for havoc:

$$\mathsf{WP}\ [\![\mathsf{havoc}\, x]\!]\ \psi = \exists x : \psi$$

- Practically, all information about x is lost, except indirect relations remain (after the assignment):

$$\mathsf{WP}\ [\![\mathsf{havoc}\, x]\!]\ (y = x \wedge x = z) \implies (y = z)$$

Pre-Condition of Havoc

- Concrete semantics:

$$[\![\text{havoc}\, x]\!]\, S = \{\sigma[x \mapsto z] \mid \sigma \in S,\ z \in \mathbb{Z}\}$$

- WP for havoc:

$$\text{WP}\ [\![\text{havoc}\, x]\!]\ \psi = \forall x : \psi$$

- We need ψ to hold for all values of x. Usually, we have assumes after havoc, so a typical example is

$$\text{WP}\ [\![\text{havoc}\, x]\!]\ ((y = x) \rightarrow (x = z)) \implies (y = z)$$

Pre-Condition of Havoc

- Concrete semantics:

$$[\![\text{havoc}\, x]\!]\, S = \{\sigma[x \mapsto z] \mid \sigma \in S,\ z \in \mathbb{Z}\}$$

- WP for havoc:

$$\text{WP}\ [\![\text{havoc}\, x]\!]\ \psi = \psi[x'/x] \qquad x' \text{ is fresh!}$$

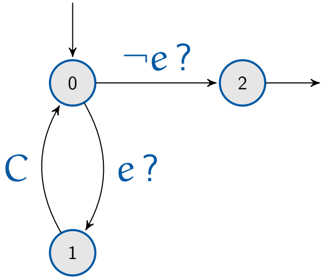
- We need ψ to hold for all values of x. Usually, we have assumes after havoc, so a typical example is

$$\text{WP}\ [\![\text{havoc}\, x]\!]\ ((y = x) \rightarrow (x = z)) \implies (y = z)$$

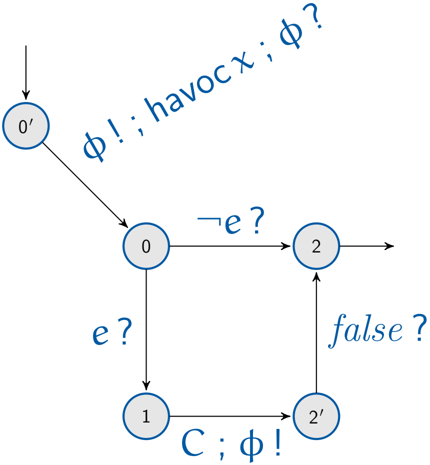
A simple assumption

- We should havoc all variables that are assigned to in the loop body.
- For simplicity, we assume this is only x.
- (You may think of x as a vector.)

Normal While Loop



Abstraction using invariant ϕ



Why can we do this?

- The construction guarantees that if

$$\bot \notin S_2$$

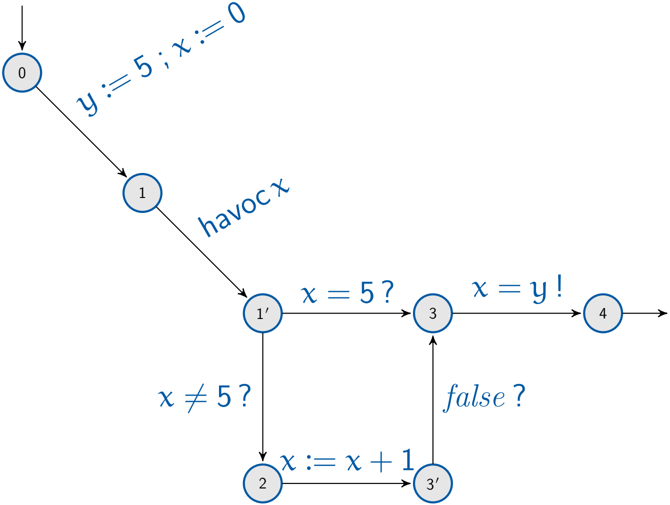
we have

$$S_2' \subseteq S_2$$

where S_i' are the sets computed for the original while loop.

- Note: it follows very closely the proof rules of Hoare logic.

Now we really can compute a VC



What happened?

- Well, there was no invariant to check.
- That's good because the invariant was trivial.
- The homework requires making this construction with an invariant.

- Just a note on procedure, and then we prove the soundness of the construction.

Procedure Calls

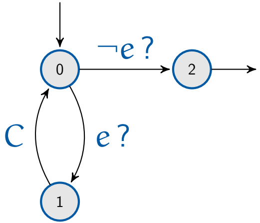
- Given a function P with parameter p and result r and contract

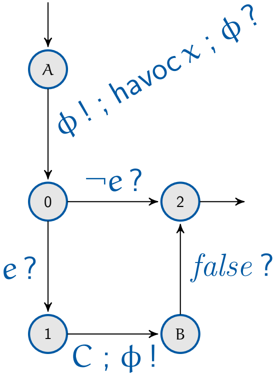
$$(\!|\, \phi \,|\!)\ P\ (\!|\, \psi \,|\!)$$

- We produce the following translation for a call $x = P(e)$.

$$\begin{aligned} &p := e \\ &\phi\,! \\ &\psi\,? \\ &x := r \end{aligned}$$

Soundness of the transformation



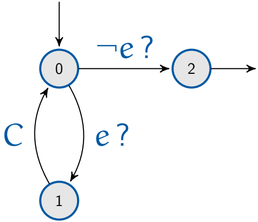


Proof Plan

1. Write down constraint systems S and S'.
2. Separate assertions into
   - the conditions they impose
   - constraint system for values
3. Show that the value system satisfies the constraints of S.
4. This implies that any solution of S' is greater than the least solution of S.

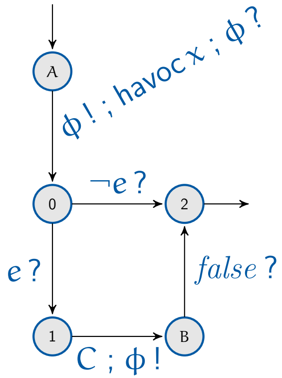
Constraint System S

$$S_0 \supseteq S$$
$$S_0 \supseteq [\![C]\!]\, S_1$$
$$S_1 \supseteq [\![e\,?]\!]\, S_0$$
$$S_2 \supseteq [\![\neg e\,?]\!]\, S_0$$



Constraint System S'

$$
\begin{aligned}
S'_A &\supseteq S \\
S'_0 &\supseteq [\![\phi\,?]\!]\,\{\sigma[x \mapsto z] \mid z \in \mathbb{Z},\\
&\qquad\qquad\qquad\qquad \sigma \in [\![\phi\,!]\!]\,S'_A\} \\
S'_1 &\supseteq [\![e\,?]\!]\,S'_0 \\
S'_B &\supseteq [\![\phi\,!]\!]\,([\![C]\!]\,S'_1) \\
S'_2 &\supseteq [\![\neg e\,?]\!]\,S'_0 \cup \{\bot \mid \bot \in S'_B\}
\end{aligned}
$$



Splitting S' based on $\bot \in S'_2$

- We can be sure $\bot \notin S'_2$ if we have

$$\begin{aligned} S &\models \phi \\ [\![C]\!]\, S'_1 &\models \phi \end{aligned}$$

- Letting $S_x = \{\sigma[x \mapsto z] \mid z \in \mathbb{Z}, \sigma \in S\}$, the following constraints remain:

$$\begin{aligned} S'_0 &\supseteq [\![\phi\,?]\!]\, S_x \\ S'_1 &\supseteq [\![e\,?]\!]\, S'_0 \\ S'_2 &\supseteq [\![\neg e\,?]\!]\, S'_0 \end{aligned}$$

Splitting S' based on $\bot \in S'_2$

- We can be sure $\bot \notin S'_2$ if we have

$$S \models \phi$$
$$[\![C]\!]\, S'_1 \models \phi$$

- Letting $S_x = \{\sigma[x \mapsto z] \mid z \in \mathbb{Z}, \sigma \in S\}$, we obtain the following solution:

$$S'_0 = \{\sigma \in S_x \mid \sigma \models \phi\}$$
$$S'_1 = \{\sigma \in S_x \mid \sigma \models \phi \wedge e\}$$
$$S'_2 = \{\sigma \in S_x \mid \sigma \models \phi \wedge \neg e\}$$

Solution to original system?

- Given the solution and conditions:

$$S'_0 = \{\sigma \in S_x \mid \sigma \models \phi\} \qquad S \models \phi$$
$$S'_1 = \{\sigma \in S_x \mid \sigma \models \phi \wedge e\} \qquad [\![C]\!]\, S'_1 \models \phi$$
$$S'_2 = \{\sigma \in S_x \mid \sigma \models \phi \wedge \neg e\}$$

- We check if the original constraints are satisfied:

$$S'_0 \supseteq S \qquad S'_0 \supseteq [\![C]\!]\, S'_1$$
$$S'_1 \supseteq [\![e\,?]\!]\, S'_0 \qquad S'_2 \supseteq [\![\neg e\,?]\!]\, S'_0$$

What did we just do?

- We had two systems:

$$X \supseteq F(X)$$
$$X \supseteq F'(X)$$

- We showed that for any Y

$$Y \supseteq F'(Y) \implies Y \supseteq F(Y)$$

- What did we conclude?

Data Flow Analysis

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

Data Flow Analysis

- We now consider how to check assertions using data flow analysis.
- Before we do that, we must to understand the basics of classical data flow analysis frameworks.

- We need to reason about soundness.
- Statements about programs are ordered…

Partial Orders

Definition

A set $\mathbb{D}$ together with a relation $\sqsubseteq$ is a partial order if for all $a, b, c \in \mathbb{D}$,

$$a \sqsubseteq a \qquad \text{reflexivity}$$

$$a \sqsubseteq b \land b \sqsubseteq a \implies a = b \qquad \text{anti-symmetry}$$

$$a \sqsubseteq b \land b \sqsubseteq c \implies a \sqsubseteq c \qquad \text{transitivity}$$

Examples

1. $\mathbb{D} = 2^{\{a,b,c\}}$ with the relation "$\subseteq$"
2. $\mathbb{Z}$ with the relation "$=$"
3. $\mathbb{Z}$ with the relation "$\leqslant$"
4. $\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp\}$ with the ordering:

$$x \sqsubseteq y \iff (x = \perp) \vee (x = y)$$

Facts about the program

- Our domain elements represent propositions about the program.
- Let $p \models x$ denote "x holds whenever execution reaches program point p".
- We order these propositions such that

$$x \sqsubseteq y \text{ whenever } (p \models x) \implies (p \models y)$$

- Consider examples:
  - The set of possibly live variables.
  - The set of definitely initialized variables.

Combining information

- Assume there are two paths to reach p (true-branch and false-branch).
- If we have x along one path and y along the other, how can we combine this information?

$$x \sqcup y$$

- We want something that is true of both paths, and
- as precise as possible.

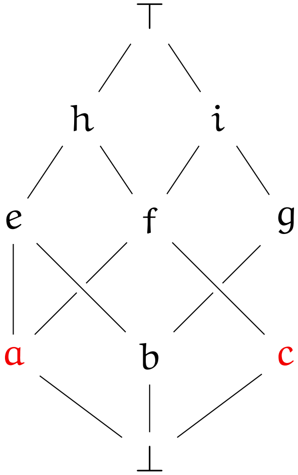
Least Upper Bounds

- $d \in \mathbb{D}$ is called an upper bound for $X \subseteq \mathbb{D}$ if

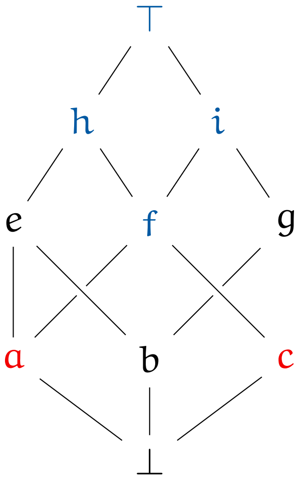
$$x \sqsubseteq d \qquad \text{for all } x \in X$$

- d is called a least upper bound if
  1. d is an upper bound and
  2. $d \sqsubseteq y$ for every upper bound y of X.

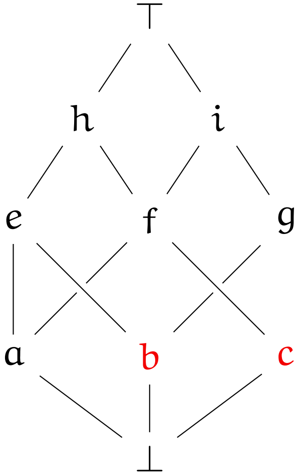
Do least upper bounds always exist?



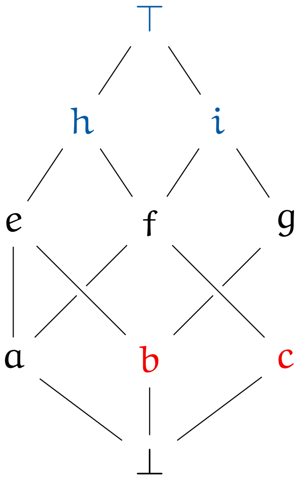
Do least upper bounds always exist?



Do least upper bounds always exist?



Do least upper bounds always exist?



Complete Lattice

Definition

A complete lattice $\mathbb{D}$ is a partial ordering where every subset $X \subseteq \mathbb{D}$ has a least upper bound $\bigsqcup X \in \mathbb{D}$.

Every complete lattice has

- a least element $\bot = \bigsqcup \emptyset \in \mathbb{D}$;
- a greatest element $\top = \bigsqcup \mathbb{D} \in \mathbb{D}$.

Which are complete lattices?

1. $\mathbb{D} = 2^{\{a,b,c\}}$
2. $\mathbb{D} = \mathbb{Z}$ with "$=$".
3. $\mathbb{D} = \mathbb{Z}$ with "$\leqslant$".
4. $\mathbb{D} = \mathbb{Z}_\perp$.

Which are complete lattices?

1. $\mathbb{D} = 2^{\{a,b,c\}}$
2. $\mathbb{D} = \mathbb{Z}$ with "$=$".
3. $\mathbb{D} = \mathbb{Z}$ with "$\leqslant$".
4. $\mathbb{D} = \mathbb{Z}_\perp$.
5. $\mathbb{Z}_\perp^\top = \mathbb{Z} \cup \{\perp, \top\}$.

Proof demo: Greatest Lower Bounds

Recall the definition

A complete lattice $\mathbb{D}$ is a partial ordering where every subset $X \subseteq \mathbb{D}$ has a least upper bound $\bigsqcup X \in \mathbb{D}$.

Theorem

If $\mathbb{D}$ is a complete lattice, then every subset $X \subseteq \mathbb{D}$ has a greatest lower bound $\bigsqcap X$.

Proof

- $L = \{l \mid \forall x \in X : l \sqsubseteq x\}$.
- Let $g = \bigsqcup L$.
- (Least upper bound of the lower bounds.)

- We show that $g = \bigsqcap X$.
    1. Show that g is a lower bound of X.
    2. Show that g is the greatest lower bound.

Solving constraint systems

- Recall the concrete semantics:

$$S_q \supseteq [\![c]\!]\, S_p \qquad \text{for } (p, c, q) \in E$$

- In general:

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n)$$

- We rewrite multiple constraints:

$$x \sqsupseteq d_1 \wedge \cdots \wedge x \sqsupseteq d_k \iff x \sqsupseteq \bigsqcup \{d_1, \ldots, d_k\}$$

So how to do it?

- In order to solve:

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n)$$

- We need f_i to be monotonic.
- A mapping f is monotonic if

$$a \sqsubseteq b \implies f(a) \sqsubseteq f(b)$$

Monotonicity

- A mapping f is monotonic if

$$a \sqsubseteq b \implies f(a) \sqsubseteq f(b)$$

- Which of the following is not monotonic?

$$\text{inc } x = x + 1 \qquad \text{dec } x = x - 1$$

Monotonicity

- A mapping f is monotonic if

$$a \sqsubseteq b \implies f(a) \sqsubseteq f(b)$$

- Which of the following is not monotonic?

$$\begin{aligned} \text{inc } x &= x + 1 & \text{dec } x &= x - 1 \\ \text{top } x &= \top & \text{bot } x &= \bot \end{aligned}$$

Monotonicity

- A mapping f is monotonic if

$$a \sqsubseteq b \implies f(a) \sqsubseteq f(b)$$

- Which of the following is not monotonic?

$$\begin{array}{ll} \text{inc } x = x + 1 & \text{dec } x = x - 1 \\ \text{top } x = \top & \text{bot } x = \bot \\ \text{id } x = x & \text{inv } x = -x \end{array}$$

Vector function

- We want to solve:

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n)$$

- Construct vector function $F \colon D^n \to D^n$

$$F(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$$

where $y_i = f_i(x_1, \ldots, x_n)$

- If f_i are monotonic, so is F.

Kleene iteration

- Successively iterate from $\bot$:

$$\bot, \quad F(\bot), \quad F^2(\bot), \quad \ldots$$

- Stop if we reach some $X = F^n(\bot)$ with

$$F(X) = X$$

- Will this terminate?
- Is this the least solution?

Simple Example

- For $\mathbb{D} = 2^{\{a,b,c\}}$

$$
\begin{aligned}
x_1 &\sqsupseteq \{a\} \cup x_3 \\
x_2 &\sqsupseteq x_3 \cap \{a, b\} \\
x_3 &\sqsupseteq x_1 \cup \{c\}
\end{aligned}
$$

- The Iteration

	0	1	2	3	4
x_1	$\emptyset$				
x_2	$\emptyset$				
x_3	$\emptyset$				

Simple Example

- For $\mathbb{D} = 2^{\{a,b,c\}}$

$$
\begin{aligned}
x_1 &\sqsupseteq \{a\} \cup x_3 \\
x_2 &\sqsupseteq x_3 \cap \{a, b\} \\
x_3 &\sqsupseteq x_1 \cup \{c\}
\end{aligned}
$$

- The Iteration

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$			
x_2	$\emptyset$	$\emptyset$			
x_3	$\emptyset$	$\{c\}$			

Simple Example

- For $\mathbb{D} = 2^{\{a,b,c\}}$

$$x_1 \sqsupseteq \{a\} \cup x_3$$
$$x_2 \sqsupseteq x_3 \cap \{a, b\}$$
$$x_3 \sqsupseteq x_1 \cup \{c\}$$

- The Iteration

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$		
x_2	$\emptyset$	$\emptyset$	$\emptyset$		
x_3	$\emptyset$	$\{c\}$	$\{a, c\}$		

Simple Example

- For $\mathbb{D} = 2^{\{a,b,c\}}$

$$x_1 \sqsupseteq \{a\} \cup x_3$$
$$x_2 \sqsupseteq x_3 \cap \{a, b\}$$
$$x_3 \sqsupseteq x_1 \cup \{c\}$$

- The Iteration

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{a, c\}$	
x_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	
x_3	$\emptyset$	$\{c\}$	$\{a, c\}$	$\{a, c\}$	

Simple Example

- For $\mathbb{D} = 2^{\{a,b,c\}}$

$$x_1 \sqsupseteq \{a\} \cup x_3$$
$$x_2 \sqsupseteq x_3 \cap \{a, b\}$$
$$x_3 \sqsupseteq x_1 \cup \{c\}$$

- The Iteration

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{a, c\}$	✓
x_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	✓
x_3	$\emptyset$	$\{c\}$	$\{a, c\}$	$\{a, c\}$	✓

Why Kleene iteration works

1. $\bot, F(\bot), F^2(\bot), \ldots$ is an ascending chain

$$\bot \sqsubseteq F(\bot) \sqsubseteq F^2(\bot) \sqsubseteq \cdots$$

2. If $F^k(\bot) = F^{k+1}(\bot)$, it is the least solution.
3. If all ascending chains in $\mathbb{D}$ are finite, Kleene iteration terminates.

Discussion

- What if $\mathbb{D}$ does contain infinite ascending chains?
- In particular, our concrete semantics was defined as the set of states with $\sigma \in V \to \mathbb{N}$.
- How do we know there aren't better solutions to the constraint system?

$$x = f(x) \qquad\qquad x \sqsupseteq f(x)$$

Answer to the first question

Theorem (Knaster-Tarski)

Assume $\mathbb{D}$ is a complete lattice. Then every monotonic function $f\colon \mathbb{D} \to \mathbb{D}$ has a least fixpoint $d_0 \in \mathbb{D}$ where

$$d_0 = \bigsqcap P \qquad\qquad P = \{d \in \mathbb{D} \mid d \sqsupseteq f(d)\}$$

1. Show that $d_0 \in P$.
2. Show that d_0 is a fixpoint.
3. Show that d_0 is the least fixpoint.

Answer to the second question

- Could there be better solutions to the constraint system than the least fixpoint?
- According to the theorem:

$$d_0 = \bigsqcap\{d \in \mathbb{D} \mid d \sqsupseteq f(d)\}$$

- Thus, d_0 is a lower bound for all solutions to the constraint system $d \sqsupseteq f(d)$.

Chaotic iteration

1. Set all x_i to $\bot$ and $W = \{1, \ldots, n\}$.
2. Take some $i \in W$ out of W.
   (if $W = \emptyset$, exit).
3. Compute $n := f_i(x_1, \ldots, x_n)$.
4. If $x_i \sqsupseteq n$, goto 2.
5. Set $x_i := x_i \sqcup n$ and reset $W := \{1, \ldots, n\}$.
6. Goto 2.

Data flow versus paths

- We want to verify that "whenever execution reaches program point p, a certain assertion holds."
- We need to check every path leading to p.
- Then: Why are we solving data flow constraint systems??

Path Semantics

- We define a path π inductively:

$$\begin{aligned} \pi &= \epsilon && \text{empty path} \\ \pi &= \pi' e && \text{where } e \in E \end{aligned}$$

- If π is a path from p to q, we write $\pi\colon p \to q$.

- We define the path semantics:

$$\begin{aligned} [\![\epsilon]\!]\, S &= S \\ [\![\pi(p, c, q)]\!]\, S &= [\![c]\!]\, ([\![\pi]\!]\, S) \end{aligned}$$

Merge Over All Paths

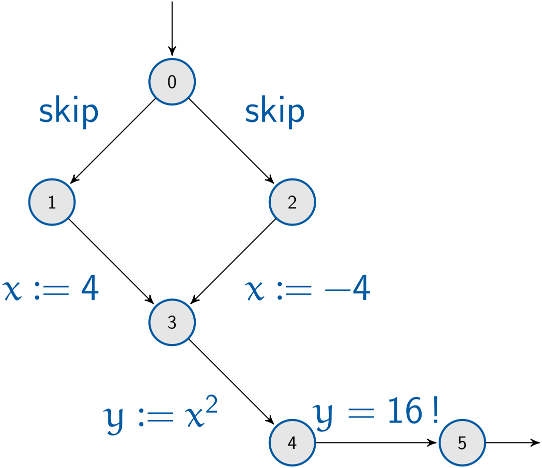
- For a complete lattice $\mathbb{D}$, we solved

$$
\begin{aligned}
x_s &\sqsupseteq d_s \\
x_q &\sqsupseteq [\![c]\!]\, x_p \quad (p, c, q) \in E
\end{aligned}
$$

- But we are really interested in:

$$y_p = \bigsqcup \{[\![\pi]\!]\, d_s \mid \pi\colon s \to p\}$$

Example: Merge Over All Paths



When do solutions coincide?

- For our collecting semantics, they do.
- All functions $[\![c]\!]$ are distributive.
- In reality, we compute an abstract semantics.

$$
\begin{aligned}
x_s &\sqsupseteq d_s \\
x_q &\sqsupseteq [\![c]\!]^\sharp \, x_p \quad (p, c, q) \in E
\end{aligned}
$$

- Transfer functions $[\![c]\!]^\sharp \colon \mathbb{D} \to \mathbb{D}$ are monotonic.

Soundness of LFP Solutions

Theorem (Kam, Ullman, 1975)

Let x_i satisfy the following constraint system:

$$\begin{aligned} x_s &\sqsupseteq d_s \\ x_q &\sqsupseteq [\![c]\!]^\sharp\, x_p \quad (p, c, q) \in E \end{aligned}$$

where $[\![c]\!]^\sharp$ are monotonic. Then, for every $p \in N$, we have

$$x_p \sqsupseteq \bigsqcup \{ [\![\pi]\!]^\sharp\, d_s \mid \pi \colon s \to p \}$$

Proof

- We need to show that for each $\pi\colon s \to p$:

$$x_p \sqsupseteq [\![\pi]\!]^\sharp\, d_s$$

- By induction on the length of π (assume the above holds for all paths of length $\leqslant n$ to any node).
  - Base case.
    - There is only one zero-length path: $\pi = \epsilon$.
    - We have $x_s \sqsupseteq [\![\epsilon]\!]^\sharp\, d_s$ from the first constraint.
  - Inductive step: Let $\pi = \pi'(p, c, q)$.
    - We have $x_p \sqsupseteq [\![\pi']\!]^\sharp\, d_s$ from the inductive hypothesis.
    - We need $x_q \sqsupseteq [\![\pi]\!]^\sharp\, d_s = [\![c]\!]^\sharp\, ([\![\pi']\!]^\sharp\, d_s)$.
    - From monotonicity: $x_q \sqsupseteq [\![c]\!]^\sharp\, x_p \sqsupseteq [\![c]\!]^\sharp\, ([\![\pi']\!]^\sharp\, d_s)$.

On Distributivity

- A function $f\colon \mathbb{D}_1 \to \mathbb{D}_2$ is distributive if for all $\emptyset \neq X \subseteq \mathbb{D}_1$:

$$f\left(\bigsqcup X\right) = \bigsqcup \{f\,x \mid x \in X\}$$

- It is strict if

$$f\,\bot = \bot$$

- It is totally distributive if both distributive and strict (distributes also $\emptyset$).

Why these distinctions?

- Many useful analyses are distributive, but…
- we generally do not have strict transfer functions.
- Instead, we assume each node v is reachable from the start node.
- Under these assumptions, distributivity suffices for our coinidence theorem.

Intraprocedural Coincidence

Theorem (Kildall, 1972)

Let x_i satisfy the following constraint system:

$$\begin{aligned} x_s &\sqsupseteq d_s \\ x_q &\sqsupseteq [\![c]\!]^\sharp x_p \quad (p, c, q) \in E \end{aligned}$$

where $[\![c]\!]^\sharp$ are distributive. Then, for every $p \in N$, we have

$$x_p = \bigsqcup \{[\![\pi]\!]^\sharp d_s \mid \pi \colon s \to p\}$$

Proof I

- Note that any distributive function is also monotonic. Simple proof using:

$$x \sqsubseteq y \iff x \sqcup y = y$$

- Thus, we only need to show this direction:

$$x_p \sqsubseteq \bigsqcup \{ [\![\pi]\!]^\sharp \, d_s \mid \pi \colon s \to p \}$$

- For this, we show that the MOP solution satisfies our constraint system. (WHY?)

Proof II

- We show for an edge (p, c, q):

$$x_q \sqsupseteq [\![c]\!]^\sharp x_p$$

- We compute:

$$\begin{aligned} x_q &= \bigsqcup\{[\![\pi]\!]^\sharp d_s \mid \pi\colon s \to q\} \\ &\sqsupseteq \bigsqcup\{[\![\pi]\!]^\sharp d_s \mid \pi\colon s \to p \to q\} \\ &= \bigsqcup\{[\![c]\!]^\sharp ([\![\pi]\!]^\sharp d_s) \mid \pi\colon s \to p\} \\ &= [\![c]\!]^\sharp \left(\bigsqcup\{[\![\pi]\!]^\sharp d_s \mid \pi\colon s \to p\}\right) \\ &= [\![c]\!]^\sharp x_p \end{aligned}$$

Implementing a constraint solver

- Given the definitions:

a_s	:	$\mathbb{D}$	value at program start
$[\![s]\!]^\sharp$	:	$\mathbb{D} \to \mathbb{D}$	abstract semantics

- Solve the following system:

$x_q \sqsupseteq d_s$	q	entry point
$x_q \sqsupseteq [\![c]\!]^\sharp \, x_p$	(p, c, q)	edge

Representation of Right-Hand Sides

- For each variable $x \in V$, we have a single constraint f_x.
- Given the sets

$$V\colon \text{Constraint Variables (\textit{Unknowns})}$$
$$\mathbb{D}\colon \text{The abstract value domain.}$$

- The type of right hand sides are

$$f_x\colon (V \to \mathbb{D}) \to \mathbb{D}$$

The example encoded

- Mathematical formulation:

$$x_1 \sqsupseteq \{a\} \cup x_3$$
$$x_2 \sqsupseteq x_3 \cap \{a, b\}$$
$$x_3 \sqsupseteq x_1 \cup \{c\}$$

- Functional encoding:

$$f_{x_1} = \lambda\sigma.\ \{a\} \cup \sigma\, x_3$$
$$f_{x_2} = \lambda\sigma.\ \sigma\, x_3 \cap \{a, b\}$$
$$f_{x_3} = \lambda\sigma.\ \sigma\, x_1 \cup \{c\}$$

Encoding in Haskell

```
data V = X1 | X2 | X3 deriving (Eq,Show)

class    FSet v where vars :: [v]
instance FSet V where vars =  [X1,X2,X3]

f X1 = \σ → S.fromList ['a'] ∪ (σ X3)
f X2 = \σ → (σ X3) ∩ S.fromList ['a','b']
f X3 = \σ → (σ X1) ∪ S.fromList ['c']
```

Assignments and Solutions

- Given a variable assignment $\sigma \colon V \to \mathbb{D}$,
- we can evaluate a right-hand-side $f\,\sigma \in \mathbb{D}$.
- An assignment σ satisfies a constraint $x \sqsupseteq f_x$ iff

$$\sigma\,x \sqsupseteq f_x\,\sigma$$

- When σ satisfies all constrains, it is a solution.

Haskell Code: Check Solution

```
type RHS v d = (v → d) → d
type Sys v d = v → RHS v d
type Sol v d = v → d

verify σ f     = all verifyVar vars where
  verifyVar v = σ v ⊒ f v σ
```

Kleene Iteration

- We iterate a monotonic function starting from $\bot$:

$$\bot \sqsubseteq f\,\bot \sqsubseteq f\,(f\,\bot) \sqsubseteq \cdots \sqsubseteq f^i\bot$$

- Until (hopefully) we reach an i, such that

$$f^i\bot \sqsupseteq f^{i-1}\bot$$

Haskell Code: Domains

```
class Domain t where
  (⊑) :: t → t → Bool
  (⊔) :: t → t → t
  bot :: t

lfp :: Domain d => (d → d) → d
lfp f = stable (iterate f bot)

stable (x:fx:tl) | fx ⊑ x     = x
                 | otherwise = stable (fx:tl)
```

matt.might.net/articles/partial-orders/

```
iterate f x = x : iterate f (f x)
```

Haskell Code: Vector Function

```
instance (FSet v, Domain d) =>
                        Domain (v → d)
  where
  f ⊑ g = all (\v → f v ⊑ g v) vars
  f ⊔ g = \v → f v ⊔ g v
  bot   = \v → bot

solve f = lfp (flip f)
```

$$\mathrm{f}\colon V \to (V \to \mathbb{D}) \to \mathbb{D}$$

$$\texttt{flip}\,\mathrm{f}\colon (V \to \mathbb{D}) \to (V \to \mathbb{D})$$

Testing the Simple Solver

```
instance Ord e => Domain (Set e) where
  x ⊑ y = x ⊆ y
  x ⊔ y = x ∪ y
  bot   = empty

f X1 = \σ → S.fromList ['a'] ∪ (σ X3)
f X2 = \σ → (σ X3) ∩ S.fromList ['a','b']
f X3 = \σ → (σ X1) ∪ S.fromList ['c']

-------------------------------------------

*Simple> solve f
X1 → fromList "ac"
X2 → fromList "a"
X3 → fromList "ac"
```

Assertion Checking with Static Analysis

Vesal Vojdani

Department of Computer Science
University of Tartu

Formal Methods (2014)

Assertion Checking

- Track values of variables.
- Combine with WP computation.
- Infer invariants for loops.

Value Domains

- Characterize the possible values of variables whenever we reach program point p.
- A non-relational value domain:

$$\mathbb{D} = \mathrm{V} \to \mathbb{D}_{\mathbb{Z}}$$

- We consider two simple value domains:
  1. Kildall's constant propagation domain.
  2. The Interval Domain.

Non-relational Domains

- For a complete lattice $\mathbb{D}$ and finite set V,
- the set of functions $\mathbb{D} \to V$ with the point-wise ordering

$$f_1 \sqsubseteq f_2 \iff \forall v \in V : f_1(v) \sqsubseteq f_2(v)$$

  is also a complete lattice.
- For example: $\mathbb{D} = V \to 2^{\mathbb{Z}}$.

Abstract Evaluation

- Just like for concrete state $\sigma \in V \to \mathbb{Z}$:

$$\begin{aligned} [\![z]\!]\,\sigma &= z \\ [\![x]\!]\,\sigma &= \sigma\,x \\ [\![e_1 + e_2]\!]\,\sigma &= [\![e_1]\!]\,\sigma + [\![e_2]\!]\,\sigma \end{aligned}$$

- Now, we need abstract operators such that for $d \in \mathbb{D} = V \to \mathbb{D}_{\mathbb{Z}}$, we evaluate:

$$\begin{aligned} [\![z]\!]^\sharp\,d &= z^\sharp \\ [\![x]\!]^\sharp\,d &= d\,x \\ [\![e_1 + e_2]\!]^\sharp\,d &= [\![e_1]\!]^\sharp\,d +^\sharp [\![e_2]\!]^\sharp\,d \end{aligned}$$

What the domain must supply

1. Lattice operations.
2. Lifting of constants:

$$\forall z \in \mathbb{Z} : z^{\sharp} \in \mathbb{D}_{\mathbb{Z}}$$

3. Abstract operations:

$$\forall z_1, z_2 \in \mathbb{D}_{\mathbb{Z}} : z_1 +^{\sharp} z_2 \in \mathbb{D}_{\mathbb{Z}}$$

(not just for $+$; also unary, comparisons, logical, etc.)

Kildall's Domain

1. Lattice is the flat lattice.
2. Constants are already elements of $\mathbb{D}_{\mathbb{Z}}$:

$$z^{\sharp} = z$$

3. Operators are essentially lifted:

$$a +^{\sharp} b = \begin{cases} \bot & \text{if } a = \bot \text{ or } b = \bot \\ \top & \text{if } a = \top \text{ or } b = \top \\ a + b & \text{otherwise} \end{cases}$$

(More precise, e.g., for multiplication?)

Interval Domain

1. Lattice is $\mathbb{Z} \times \mathbb{Z}$ with $\langle l_1, u_1 \rangle \sqsubseteq \langle l_2, u_2 \rangle$ if

$$\langle l_2 \leqslant l_1 \rangle \wedge \langle u_1 \leqslant u_2 \rangle$$

2. Constants are singleton intervals:

$$z^\sharp = \langle z, z \rangle$$

3. Operators are generally defined as:

$$\langle l_1, u_1 \rangle *^\sharp \langle l_2, u_2 \rangle = \langle l, u \rangle \text{ where}$$
$$l = \min\{a * b \mid a \in \{l_1, u_1\},\ b \in \{l_2, u_2\}\}$$
$$u = \max\{a * b \mid a \in \{l_1, u_1\},\ b \in \{l_2, u_2\}\}$$

The Analysis

- We define abstract transfer functions.
- The simple ones:

$$\begin{aligned} [\![\mathsf{skip}]\!]^\sharp\, d &= d \\ [\![x := e]\!]^\sharp\, d &= d\,[x \mapsto [\![e]\!]^\sharp\, d] \end{aligned}$$

- Much like the concrete semantics:

$$\begin{aligned} [\![\mathsf{skip}]\!]\, S &= S \\ [\![x := e]\!]\, S &= \{\sigma\,[x \mapsto [\![e]\!]\,\sigma] \mid \sigma \in S\} \end{aligned}$$

The Bottom Value

- The bottom element is the mapping

$$\mathrm{d}\, v = \bot \; (\forall v \in V)$$

- As soon as $\exists v$ with $\mathrm{d}\, v = \bot$, we would set all variables to $\bot$.
- The bottom value then denotes non-reachability.
- All transfer functions would strictly let $\bot$ pass through.
- Why allow $\bot$ in the value domains at all?

Assume edges

- The concrete semantics:

$$[\![e\,?]\!]\,S = \{\sigma \mid \sigma \in S_p,\ [\![e]\!]\,\sigma \neq 0\} \cup \{\bot \mid \bot \in S_p\}$$

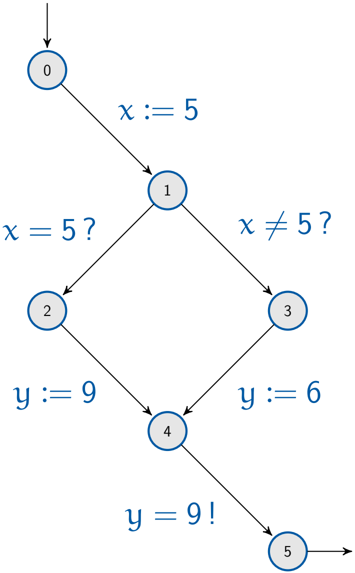
- We will handle errors separately.
- Abstract value sets:

$$[\![e\,?]\!]^\sharp\,d = \begin{cases} \bot & \text{if } [\![e]\!]^\sharp\,d = 0 \\ d \sqcap d_t & \text{otherwise} \end{cases}$$

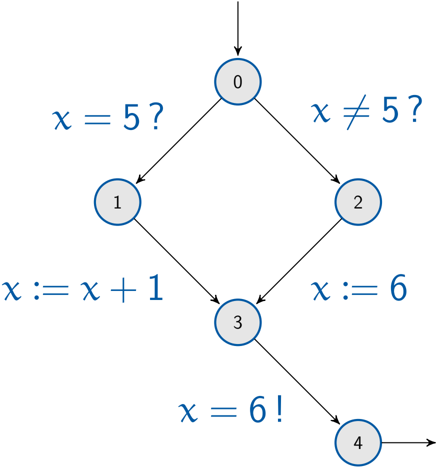
where

$$d_t = \bigsqcup \text{minimal\_elems}\{d \mid [\![e]\!]^\sharp\,d \neq 0\}$$

Example 1: Dead Code



Example 2: Restricting Values



Correctness

- We have a monotonic concretization function γ.
- For the value domains $\gamma\colon \mathbb{D}_{\mathbb{Z}} \to 2^{\mathbb{Z}}$.

$$\gamma\ z = \begin{cases} \emptyset & \text{if } a = \bot \\ \mathbb{Z} & \text{if } a = \top \\ \{z\} & \text{otherwise} \end{cases}$$

- For the variable assignments:

$$\gamma\ \mathrm{d} = \begin{cases} \emptyset & \text{if } \exists v : \mathrm{d}\, v = \bot \\ \{\rho \mid \forall v : \rho\, v \in \gamma\,(\mathrm{d}\, v)\} & \text{otherwise} \end{cases}$$

Correctness condition

- All our transfer functions need to satisfy:

$$[\![c]\!]\,(\gamma\,d) \sqsubseteq \gamma\,([\![c]\!]^\sharp\,d)$$

- Then, then the least solutions also satisfy:

$$S_p \subseteq \gamma\,x_p$$

- Because if we have $f(\gamma\,x) \sqsubseteq \gamma(f^\sharp\,x)$ and $d = f^\sharp\,d$, then

$$f(\gamma\,d) \sqsubseteq \gamma(f^\sharp\,d) = \gamma\,d$$

Assert edges

- Their effect on values is like assume:

$$\begin{aligned} [\![e\,!]\!]\, S = \{\sigma \mid \sigma \in S_p,\ [\![e]\!]\,\sigma \neq 0\} \\ \cup \{\bot \mid \sigma \in S_p,\ [\![e]\!]\,\sigma = 0\} \end{aligned}$$

- So how to check assertions? (next slide)
- Let x_p be the value analysis:

$$\begin{aligned} x_0 &\sqsupseteq d_0 \\ x_q &\sqsupseteq [\![c]\!]^\sharp\, x_p \qquad \text{for } (p, c, q) \in E \end{aligned}$$

Assertion Checking

- We can just check for each assertion edge $(p, e\,!, q)$

$$1^\sharp \sqsubseteq [\![e]\!]^\sharp x_p$$

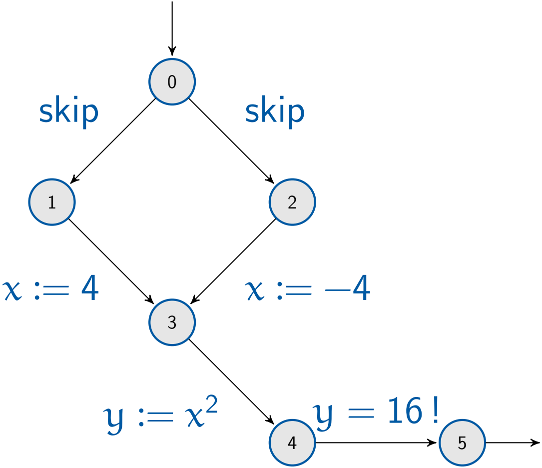
If the above does not hold, the the assertion definitely fails.

- If we want to be sound:

$$[\![e]\!]^\sharp x_p \sqsubseteq 1^\sharp$$

If this holds, the assertion is verified.

Example 3: Distributivity



Can we do better?

- We combine with WP computation.
- Recall the constraint system:

$$\phi_p \Rightarrow \text{WP}\,[\![c]\!]\,\phi_q \qquad \text{for } (p, c, q) \in E$$

- What is the ordering of the domain?

- How do we combine?
- We can set up such a system for each assertion. . .

Discussion

- It is safe if we can only approximate implication.
- What is important for soundness?
- Our domain can be sets of conjucts.
- At program point p, we can safely dismiss a conjunct ϕ if

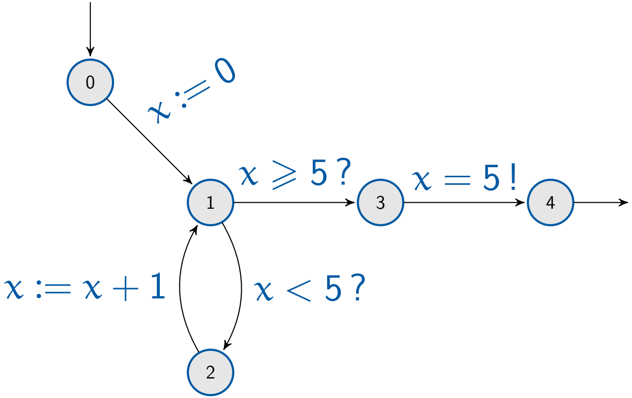
$$[\![\phi]\!]^\sharp x_p \sqsubseteq 1^\sharp$$

- If the solution for the system has $\phi_0 \equiv \mathit{true}$, we are happy.

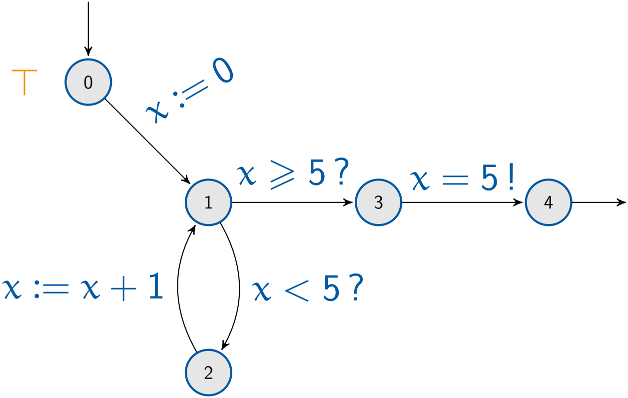
Conclusion

- This works for the simple example.
- WP computation would not terminate for a loop.
- Also, what is the concretization of this combined analysis?

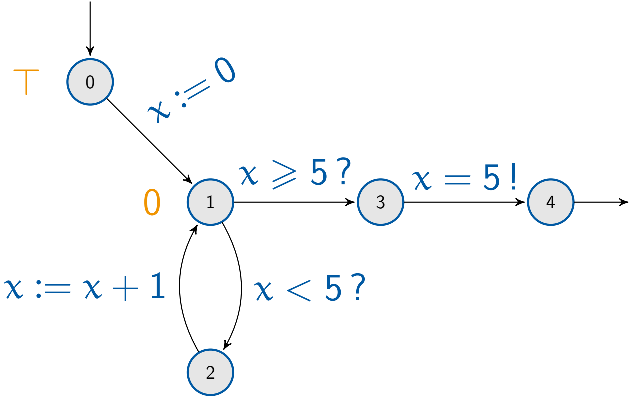
What about loops?



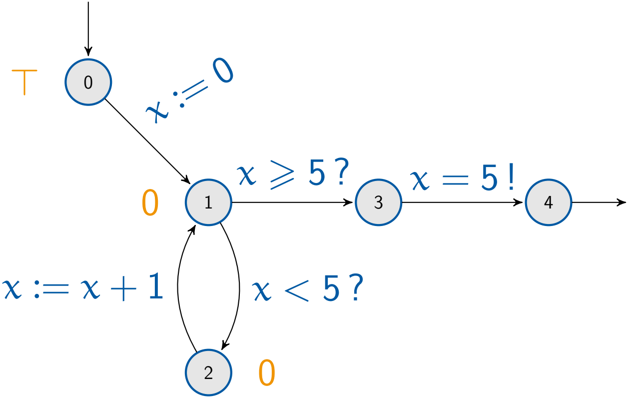
For the Kildall domain:



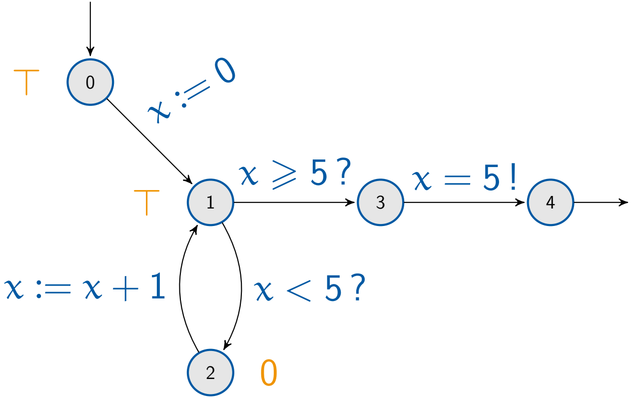
For the Kildall domain:



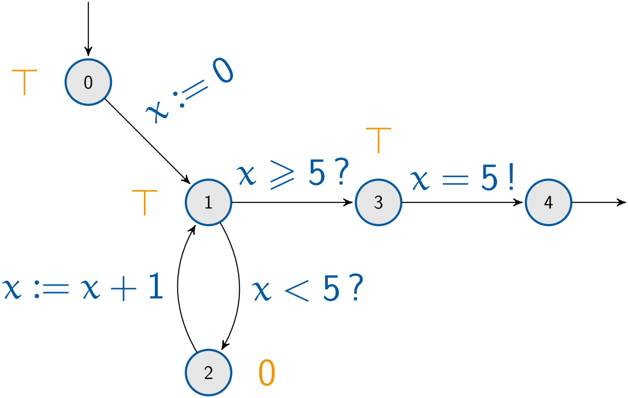
For the Kildall domain:



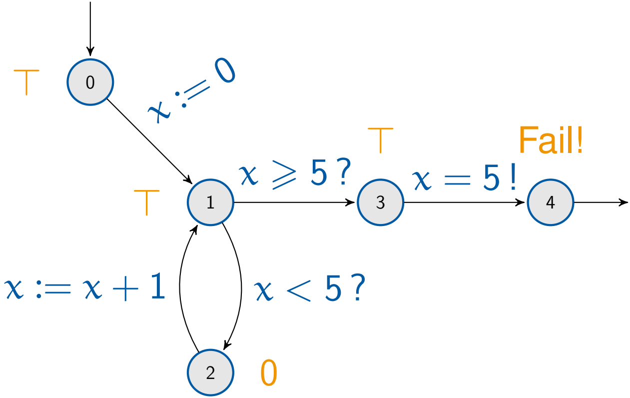
For the Kildall domain:



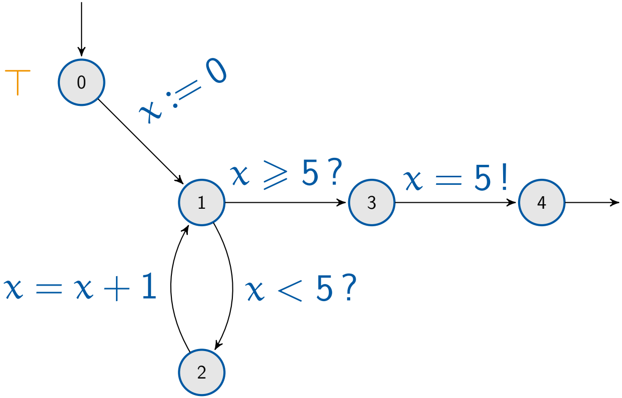
For the Kildall domain:



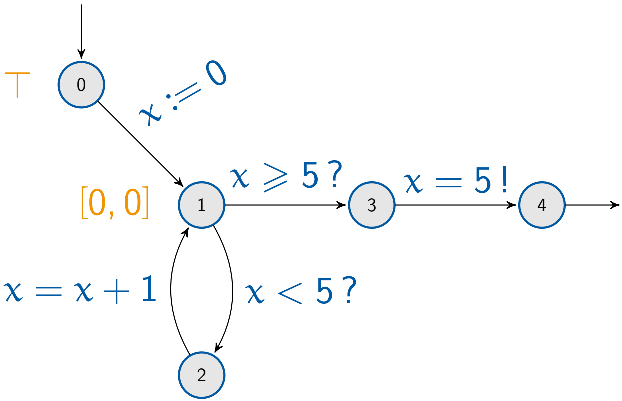
For the Kildall domain:



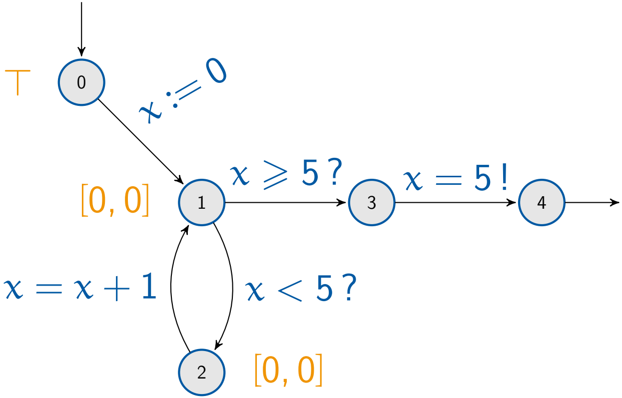
For the interval domain



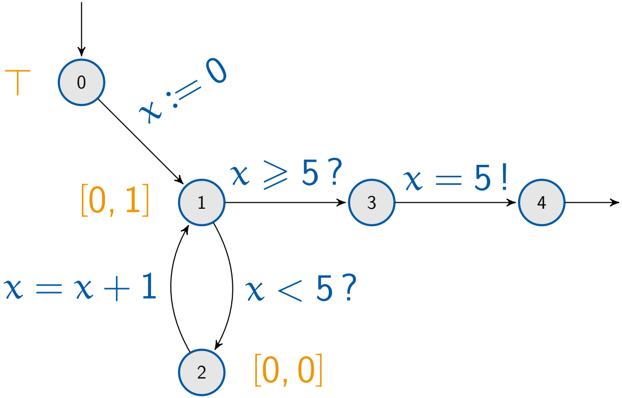
For the interval domain



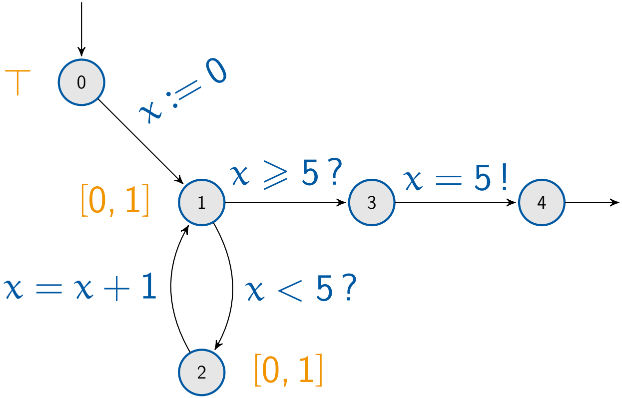
For the interval domain



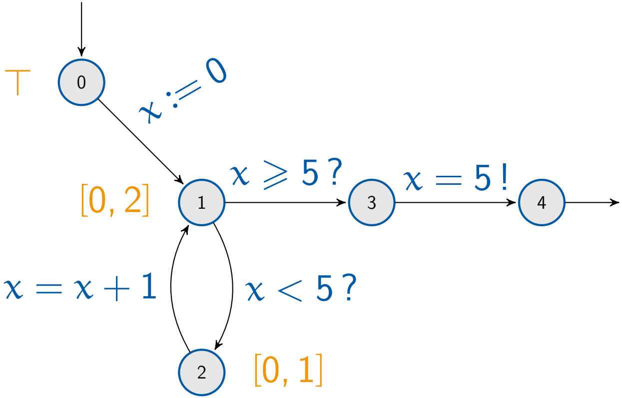
For the interval domain



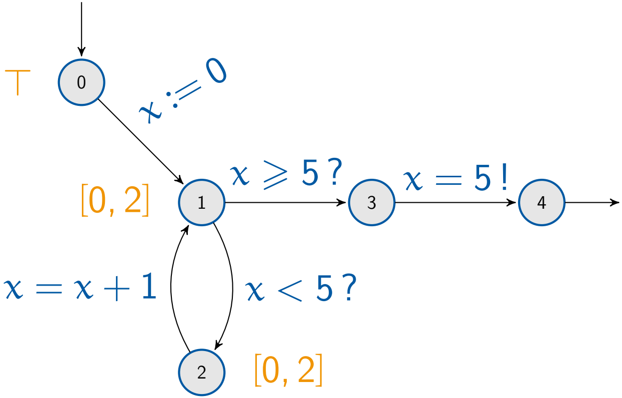
For the interval domain



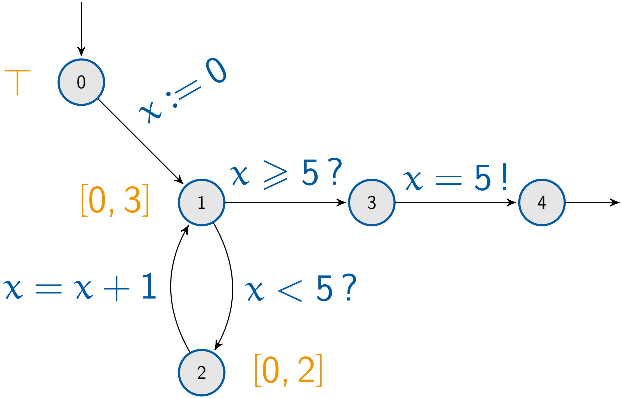
For the interval domain



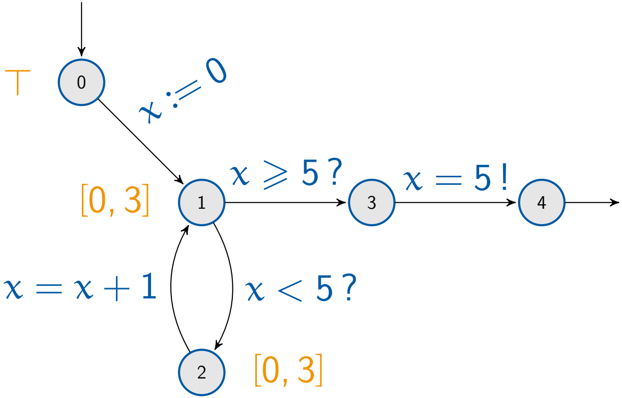
For the interval domain



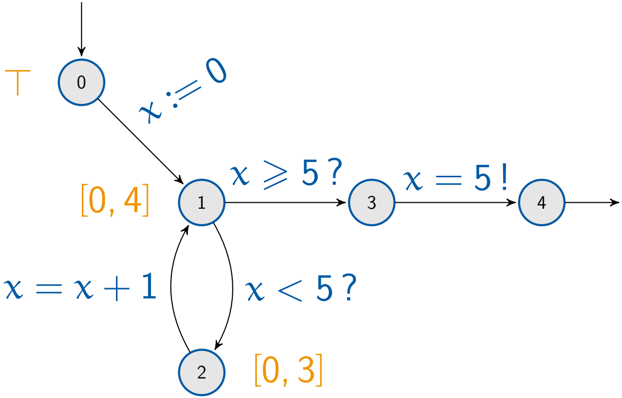
For the interval domain



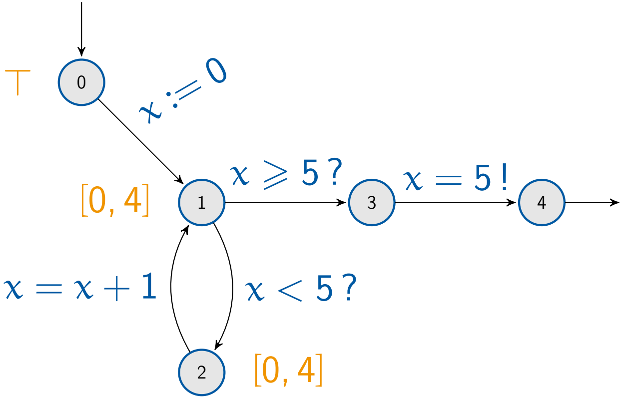
For the interval domain



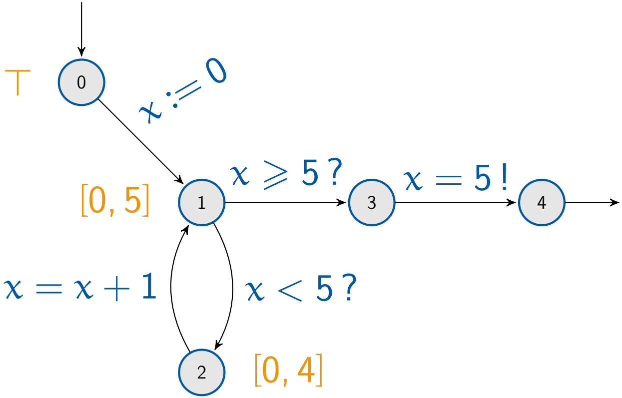
For the interval domain



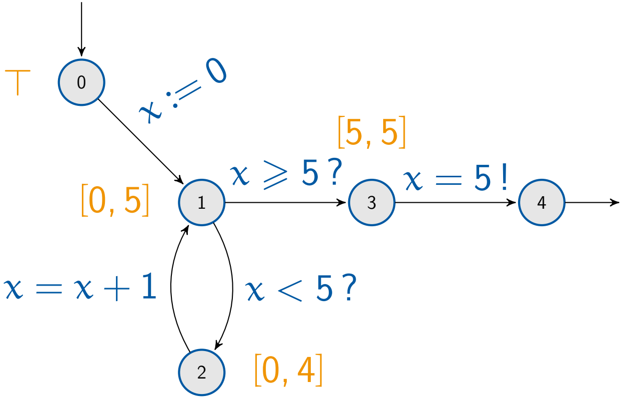
For the interval domain



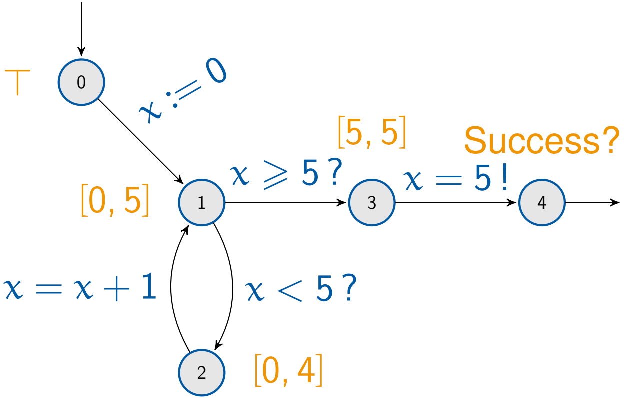
For the interval domain



For the interval domain



For the interval domain



Not really. . .

- This was not really static analysis.
- Termination not guaranteed.

- All ascending chains must stabilize.
- Enforce this by a widening operator ∇.
- Then, Kleene iteration will reach a (not necessarily least) fixpoint.

Widening

$\nabla : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ is a widening operator if

1. $\forall x, y \in \mathbb{D} : (x \sqsubseteq x \nabla y) \wedge (y \sqsubseteq x \nabla y)$
2. for every chain $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \cdots$,

$$
\begin{aligned}
y_0 &= x_0 \\
y_1 &= y_0 \nabla x_1 \\
y_2 &= y_1 \nabla x_2 \\
&\ldots
\end{aligned}
$$

is not strictly increasing.

Iteration with widening

- Our non-terminating iteration:

$$x_0 = \bot$$
$$x_{i+1} = f(x_i)$$

- Iteration with widening:

$$y_0 = \bot$$
$$y_{i+1} = \begin{cases} y_i & \text{if } f(y_i) \sqsubseteq y_i \\ y_i \nabla f(y_i) & \text{otherwise} \end{cases}$$

Widening for Intervals

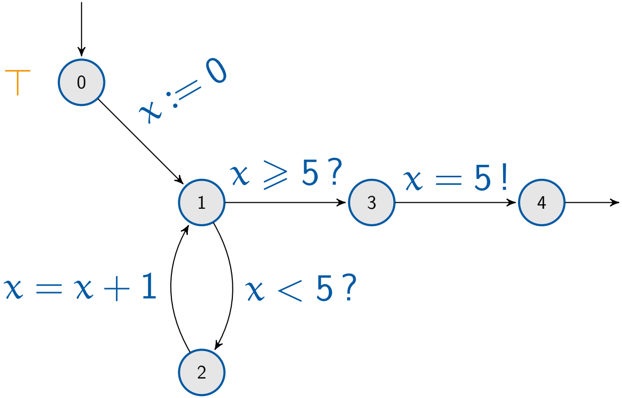
- $[l_1, u_1] \nabla [l_2, u_2] = [l, u]$ where

$$l = \begin{cases} l_1 & \text{if } l_1 \leqslant l_2 \\ -\infty & \text{otherwise} \end{cases}$$

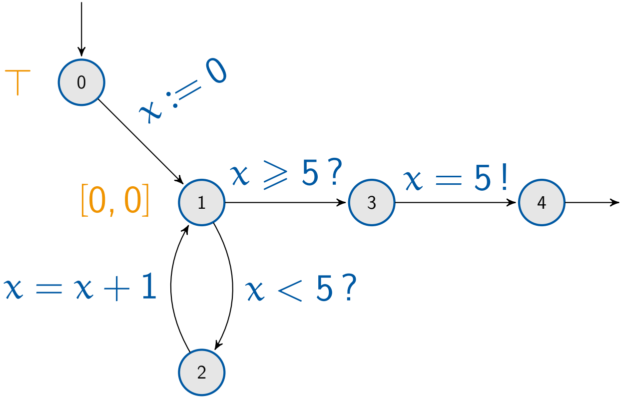
$$u = \begin{cases} u_1 & \text{if } u_2 \leqslant u_1 \\ \infty & \text{otherwise} \end{cases}$$

- This is not commutative
  - First argument: previous iteration.
  - Second argument: new value!
- Idea: give up if bounds are increasing.

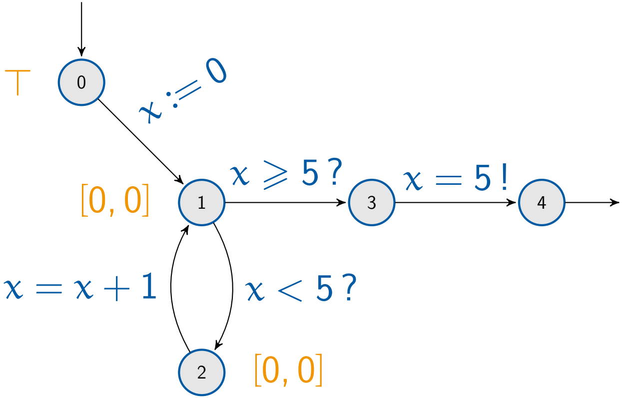
Example with widening



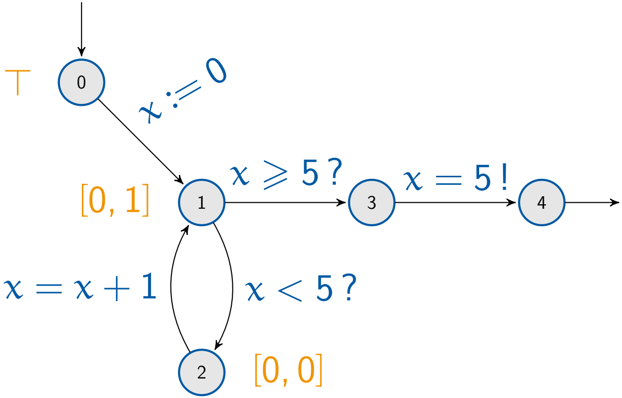
Example with widening



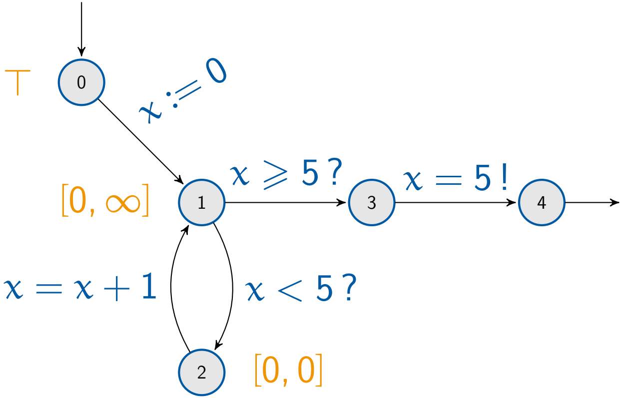
Example with widening



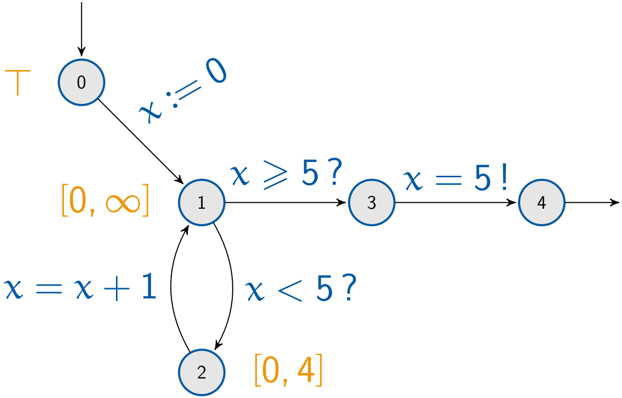
Example with widening



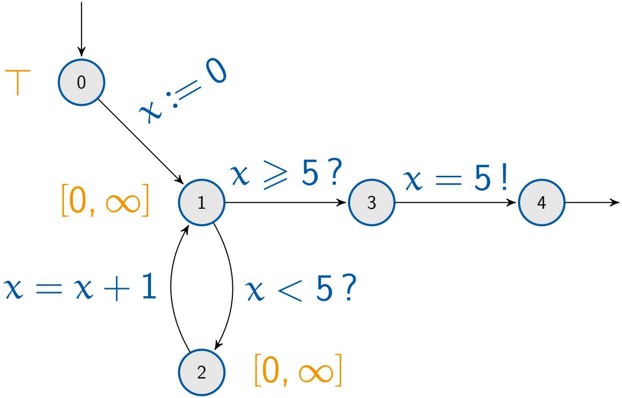
Example with widening



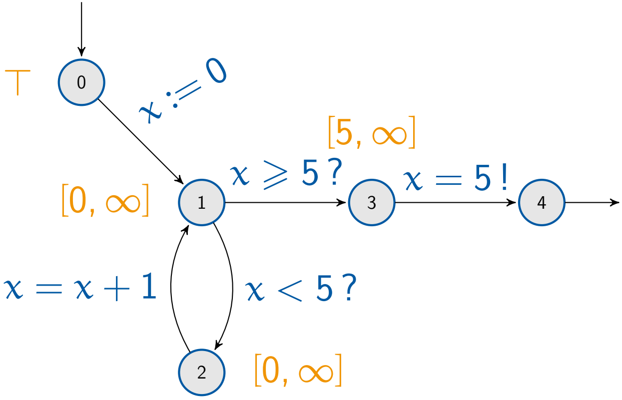
Example with widening



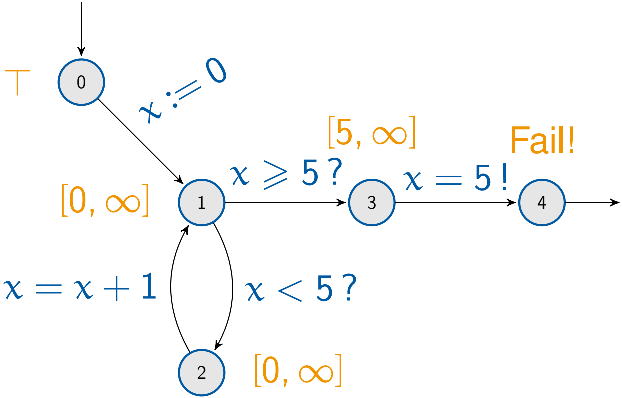
Example with widening



Example with widening



Example with widening



Why did we fail?

- We are above the least solution.
- In particular, conditional constraints are over-approximated:

$$
\begin{aligned}
x_2 &\sqsupseteq [\![x < 5\,?]\!]^\sharp\, x_1 \\
[0, \infty] &\sqsupseteq [\![x < 5\,?]\!]^\sharp\, [0, \infty] \\
[0, \infty] &\sqsupseteq [0, 4]
\end{aligned}
$$

- Idea: why not just iterate a few times more?

Refining the solution

- Let x denote a solution to our constraint system:

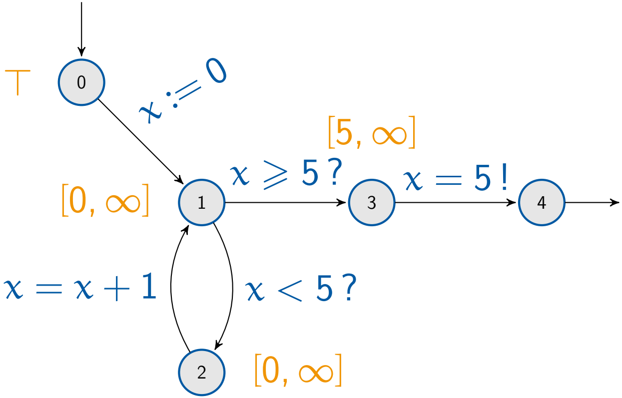
$$x \sqsupseteq f(x)$$

- If f is monotonic, then further iterations are all safe!

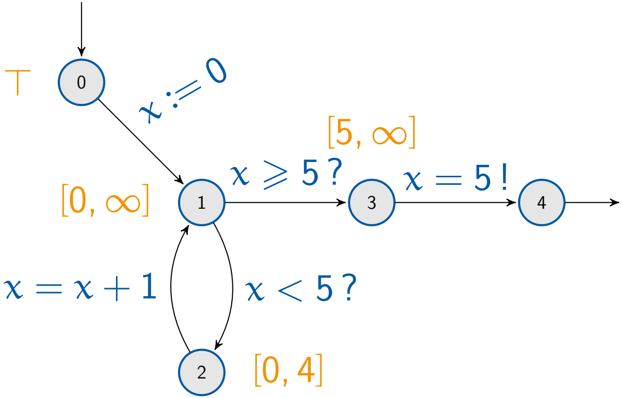
$$x \sqsupseteq f(x) \sqsupseteq f^2(x) \sqsupseteq \cdots$$

- We can stop after 5 minutes if we don't hit a fixpoint.

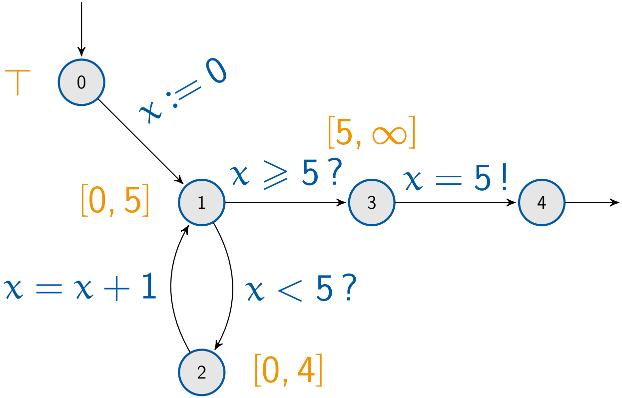
Post-fixpoint iteration



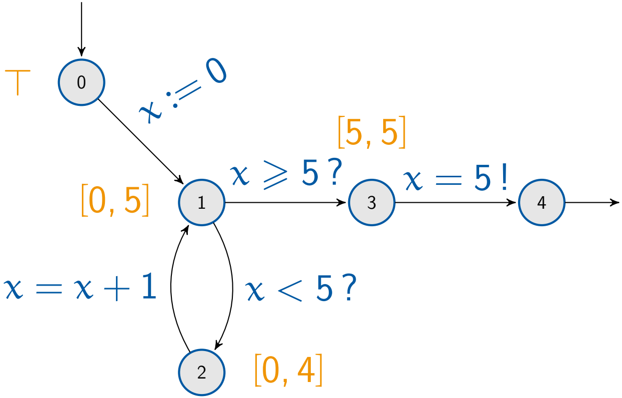
Post-fixpoint iteration



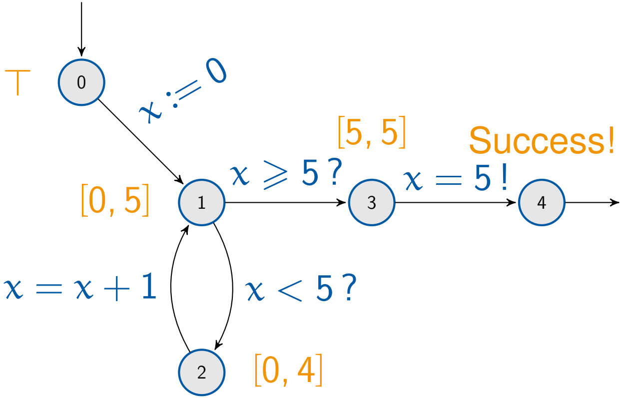
Post-fixpoint iteration



Post-fixpoint iteration



Post-fixpoint iteration



Success finally?

- Well, we were lucky and hit a fix-point.
- Termination for post-fixpoint iteration can be guaranteed.
- We require a narrowing operator $\triangle$.

Narrowing

$\triangle : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ is a narrowing operator if

1. $\forall x, y \in \mathbb{D} : (y \sqsubseteq x) \implies (y \sqsubseteq x \triangle y \sqsubseteq x)$
2. for every chain $x_0 \sqsupseteq x_1 \sqsupseteq x_2 \sqsupseteq \cdots$,

$$\begin{aligned} y_0 &= x_0 \\ y_1 &= y_0 \triangle x_1 \\ y_2 &= y_1 \triangle x_2 \\ &\cdots \end{aligned}$$

is not strictly decreasing.

Narrowing iteration

- Let x_0 be a solution, i.e.,

$$x_0 \sqsupseteq f(x_0)$$

- Post-fixpoint iteration with narrowing

$$\begin{aligned} y_0 &= x_0 \\ y_{i+1} &= y_i \triangle f(y_i) \end{aligned}$$

Narrowing for Intervals

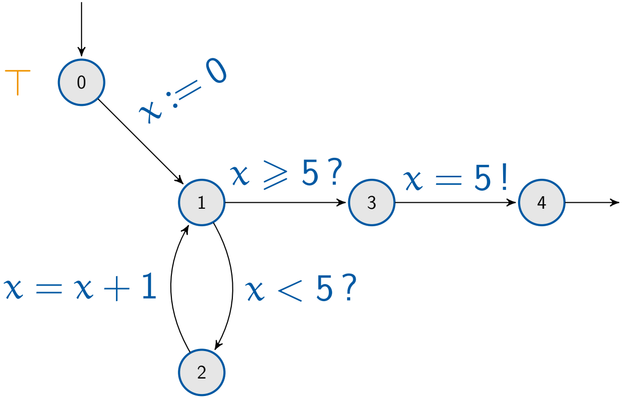
- $[l_1, u_1] \triangledown [l_2, u_2] = [l, u]$ where

$$l = \begin{cases} l_2 & \text{if } l_1 = -\infty \\ l_1 & \text{otherwise} \end{cases}$$

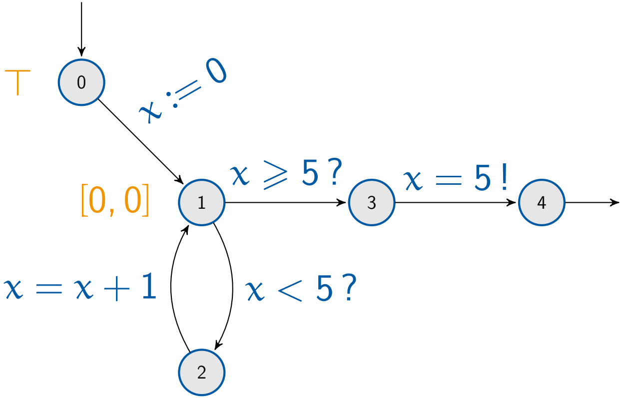
$$u = \begin{cases} u_2 & \text{if } u_1 = \infty \\ u_1 & \text{otherwise} \end{cases}$$

- Idea: Only restore lost bounds.

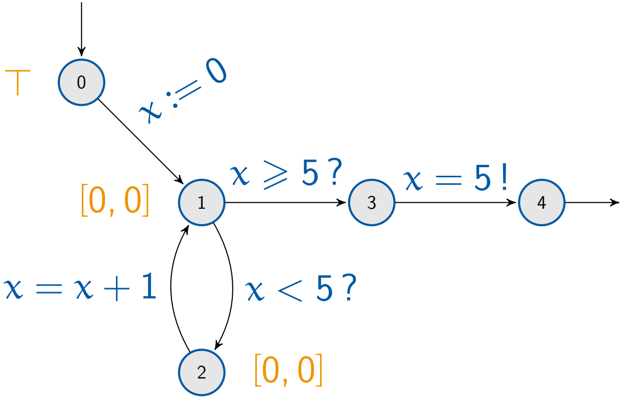
Replay with Widening/Narrowing



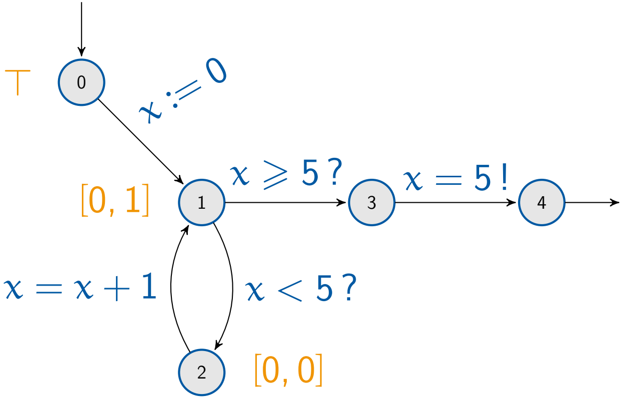
Replay with Widening/Narrowing



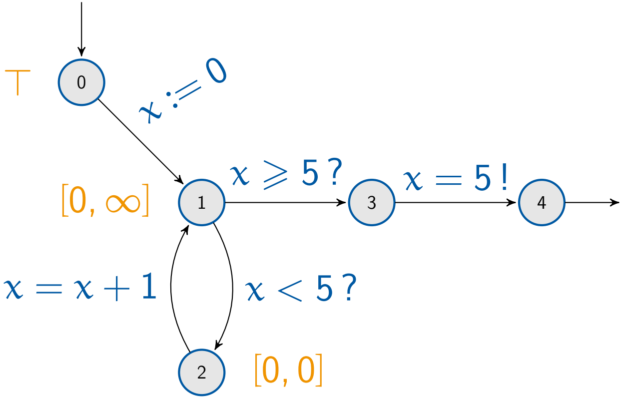
Replay with Widening/Narrowing



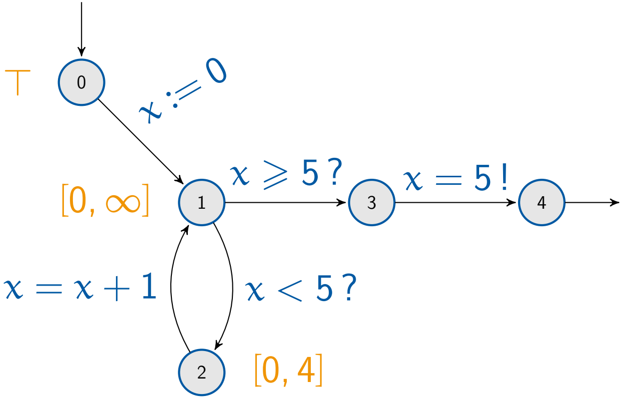
Replay with Widening/Narrowing



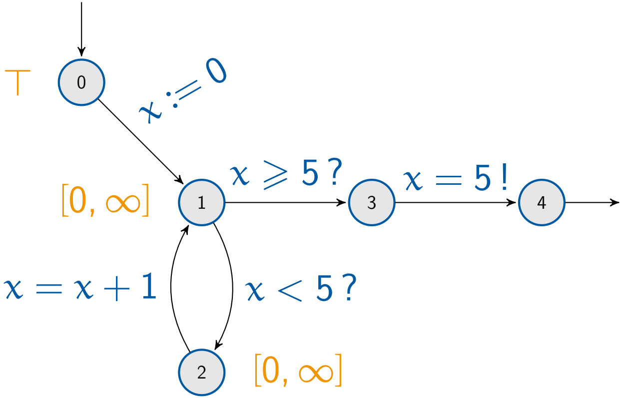
Replay with Widening/Narrowing



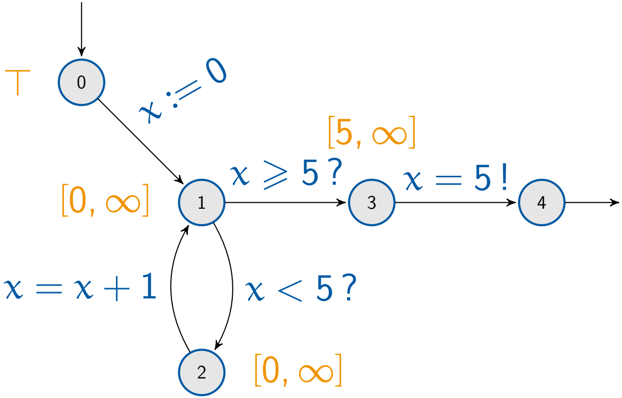
Replay with Widening/Narrowing



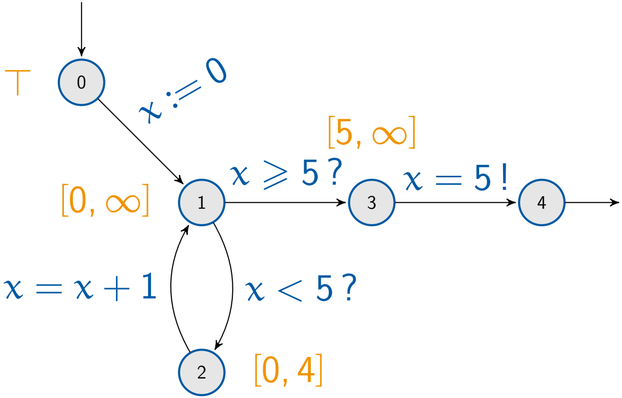
Replay with Widening/Narrowing



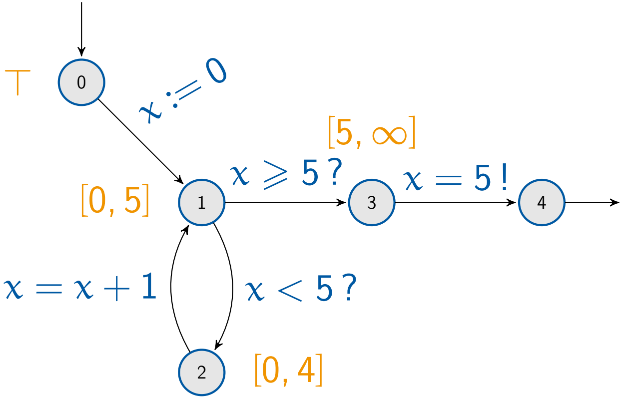
Replay with Widening/Narrowing



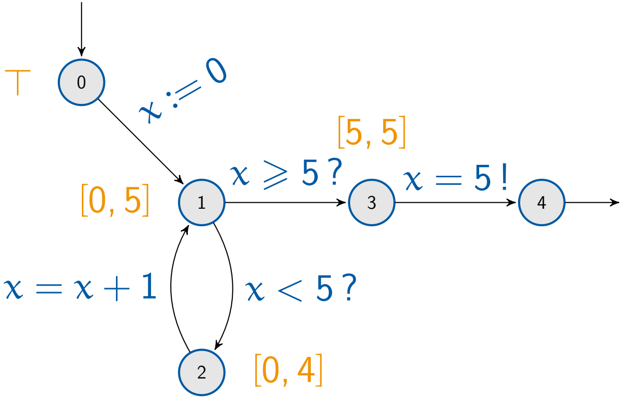
Replay with Widening/Narrowing



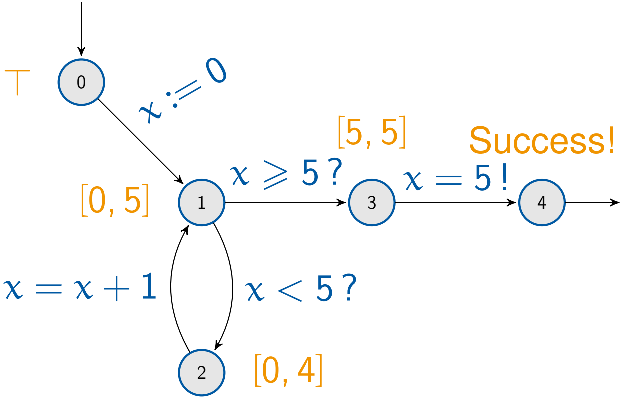
Replay with Widening/Narrowing



Replay with Widening/Narrowing



Replay with Widening/Narrowing



Conclusion

- This example does not require narrowings.
- Can you think of a simple modification to this example where narrowing would be essential?