CSCI 740 - Programming Language Theory

Lecture 29

Collecting Semantics

Instructor: Hossein Hojjat

November 8, 2017

Abstract Interpretation for Imperative Programs

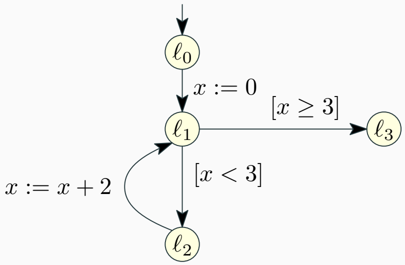
- So far we abstracted the value of expressions
- Now we want to abstract the state at each point in the program
- First we define the concrete semantics that we are abstracting
- We will use a **collecting** semantics

Collecting Semantics

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly

Example of a program annotated with its collecting semantics:

ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `      x := x + 2;`
ℓ_3 :

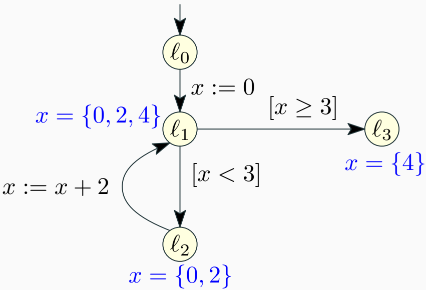


Collecting Semantics

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly

Example of a program annotated with its collecting semantics:

ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `x := x + 2;`
ℓ_3 :

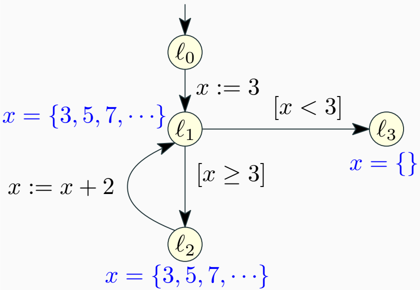


Collecting Semantics

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly

Example of a program annotated with its collecting semantics:

ℓ_0 : `int x := 3;`
ℓ_1 : `while (x >= 3)`
ℓ_2 : `    x := x + 2;`
ℓ_3 :



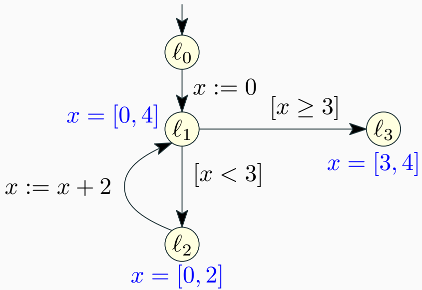
- Annotations can also be infinite sets of states

Collecting Semantics Abstraction

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly
- We can approximate sets of integers by “abstract values”
  - e.g. intervals [low,high]

Example of a program annotated with its collecting semantics:

```
ℓ0 : int x := 0;
ℓ1 : while (x < 3)
ℓ2 :       x := x + 2;
ℓ3 :
```

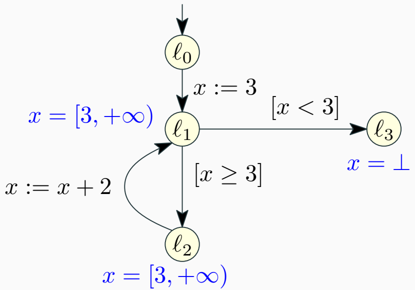


Collecting Semantics Abstraction

- Collecting semantics collects together all the states that can arise at each program point
- Neither big nor the small-step semantics express this information directly
- We can approximate sets of integers by "abstract values"
    - e.g. intervals [low,high]

Example of a program annotated with its collecting semantics:

```
ℓ0 : int x := 3;
ℓ1 : while (x >= 3)
ℓ2 :       x := x + 2;
ℓ3 :
```

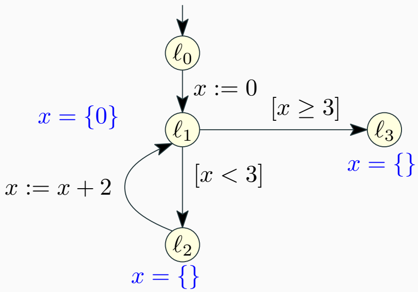


- We lose some precision but the annotations have become finitely

Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

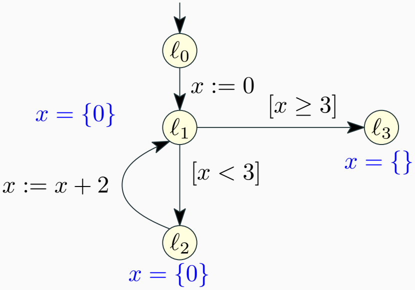
ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `x := x + 2;`
ℓ_3 :



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

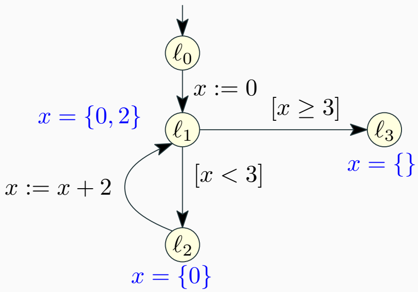
```
ℓ0 : int x := 0;
ℓ1 : while (x < 3)
ℓ2 :       x := x + 2;
ℓ3 :
```



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

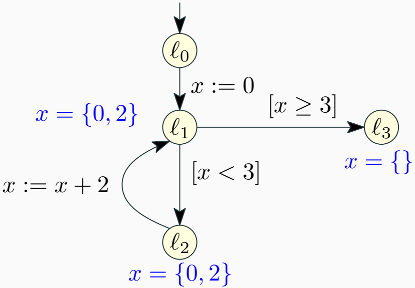
ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `      x := x + 2;`
ℓ_3 :



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

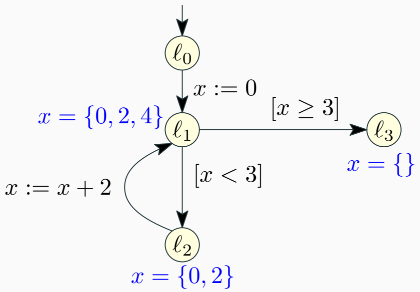
ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `x := x + 2;`
ℓ_3 :



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

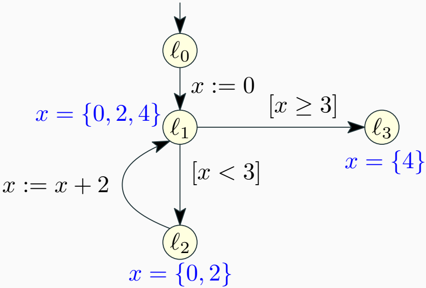
```
ℓ0 : int x := 0;
ℓ1 : while (x < 3)
ℓ2 :       x := x + 2;
ℓ3 :
```



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate execution of the program on set of values until the annotations stabilize

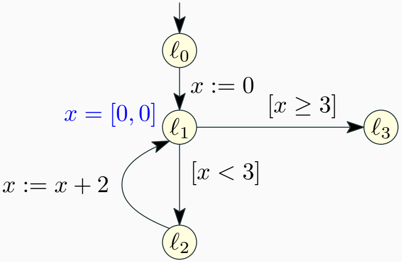
```
ℓ0 : int x := 0;
ℓ1 : while (x < 3)
ℓ2 :       x := x + 2;
ℓ3 :
```



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

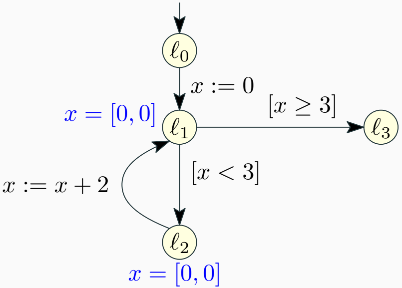
ℓ_0 : `int x := 0;`
ℓ_1 : **`while`** `(x < 3)`
ℓ_2 : `x := x + 2;`
ℓ_3 :



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

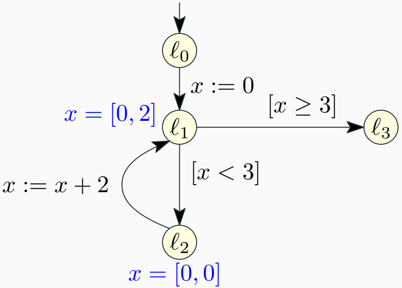
```
ℓ0 : int x := 0;
ℓ1 : while (x < 3)
ℓ2 :       x := x + 2;
ℓ3 :
```



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

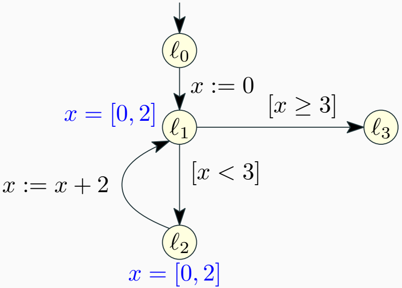
ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `x := x + 2;`
ℓ_3 :



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

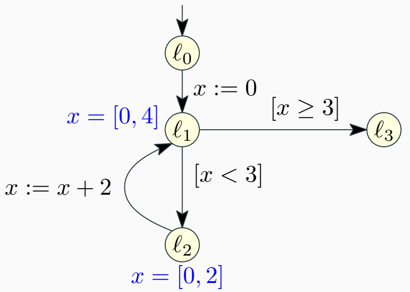
ℓ_0 : `int x := 0;`
ℓ_1 : `while (x < 3)`
ℓ_2 : `x := x + 2;`
ℓ_3 :



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

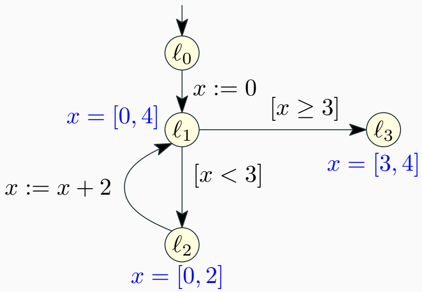
```
ℓ0 : int x := 0;
ℓ1 : while (x < 3)
ℓ2 :       x := x + 2;
ℓ3 :
```



Example: Compute Abstract Annotations

- Start from an unannotated program
- Iterate abstract execution of the program on intervals until the annotations stabilize

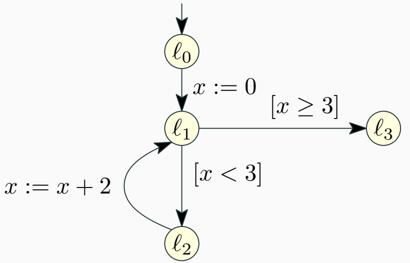
ℓ_0 :
```
int x := 0;
```
ℓ_1 :
```
while (x < 3)
```
ℓ_2 :
```
    x := x + 2;
```
ℓ_3 :



Compute Annotations

- Let $[\![\tau]\!] : \text{State} \rightarrow \text{State}$ be the transfer function of transition τ in the control-flow graph
    - (strongest post-condition)
- Let $\mathcal{X}_i$ be the set of values at i of CFG

$$\mathcal{X}_0 = \{x : \mathbb{Z}\}$$
$$\mathcal{X}_1 = [\![x := 0]\!]\mathcal{X}_0 \cup [\![x := x + 2]\!]\mathcal{X}_2$$
$$\mathcal{X}_2 = [\![x < 3]\!]\mathcal{X}_1$$
$$\mathcal{X}_3 = [\![x \geq 3]\!]\mathcal{X}_1$$



Collecting Semantics as Fixpoint

- We can associate a system of semantic equations to the collecting semantics of the form

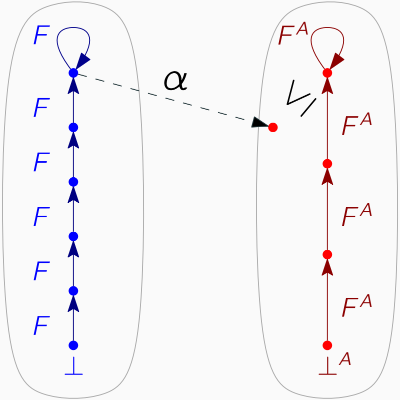
$$\mathcal{X} = F(\mathcal{X}) \quad \text{with} \quad \mathcal{X} \in (\mathcal{P}(\mathsf{Var} \to \mathbb{Z}))^n$$

- where n is the number of states in CFG
- Collecting semantics is the fixpoint solution of the semantic equation
- Tarski's theorem guarantees fixpoints in complete lattices
    - But the above proof does not say how to find them.
- It is always computable when F is omega continuous

$$\mathsf{lfp}(F) = \bigcup_{n \geq 0} F^n(\emptyset, \ldots, \emptyset)$$

Compute Abstract Annotations

We can formulate analogous constraints in abstract domain



Concrete Domain Abstract Domain

Approximate Abstraction:
$\alpha(\mathsf{lfp}(F)) \leq \mathsf{lfp}(F^A)$

Exact Abstraction:
$\alpha(\mathsf{lfp}(F)) = \mathsf{lfp}(F^A)$

Abstract Transfer Functions

For $[\![\tau]\!] : C \to C$ we define the best abstract transfer function $[\![\tau]\!]^A : A \to A$

- Let σ^A maps variables to interval lattice elements

$[\![x := y + z]\!]^A(\sigma^A) = \sigma^A[x \mapsto [l, h]]$ where $l = \sigma^A(y).\text{low} + \sigma^A(z).\text{low}$ and $l = \sigma^A(y).\text{high} + \sigma^A(z).\text{high}$

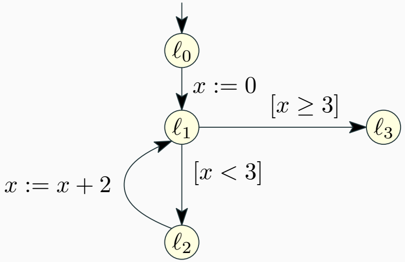
$[\![x := y + z]\!]^A(\sigma^A) = \sigma^A$ where $\sigma^A(y) = \bot \vee \sigma^A(z) = \bot$

- (Recall $[a, b] +^A [c, d] = [a + c, b + d]$)

Abstract Semantic Function

- Let $\mathcal{X}_i^A$ be the abstraction if set of values at i of CFG

$$
\begin{aligned}
\mathcal{X}_0^A &= \{x : [-\infty, +\infty]\} \\
\mathcal{X}_1^A &= [\![x := 0]\!]^A \mathcal{X}_0^A \sqcup [\![x := x + 2]\!]^A \mathcal{X}_2^A \\
\mathcal{X}_2^A &= [\![x < 3]\!]^A \mathcal{X}_1^A \\
\mathcal{X}_3^A &= [\![x \geq 3]\!]^A \mathcal{X}_1^A
\end{aligned}
$$



- Fixpoint of F^A is an over-approximate of fixpoint of F