

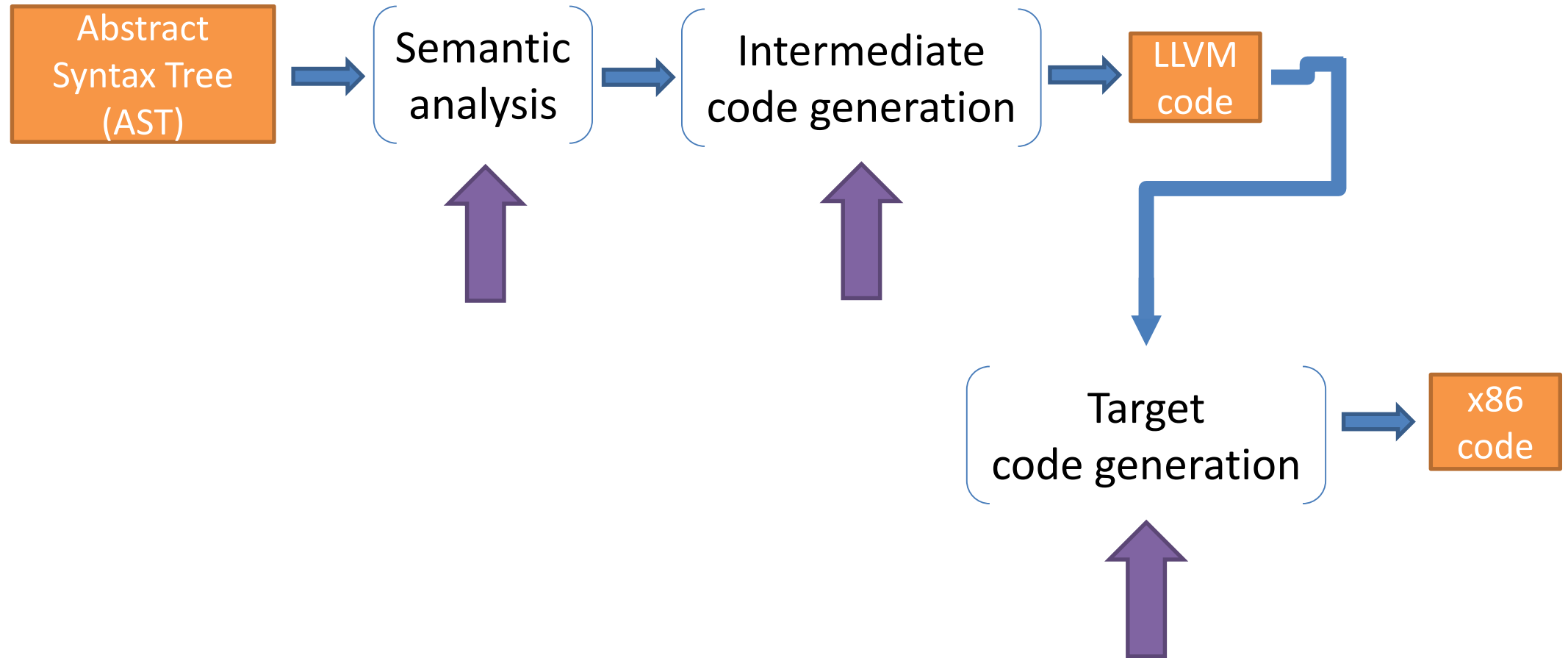
Compiler Construction

Winter 2020

Recitation 8: Static Analysis

Yotam Feldman

Semantic Analysis



Uninitialized Variables

```
int x;  
int y;  
y = x;
```

What would happen at runtime?

compilation error
(semantic analysis)

Initialized Fields

```
class A {  
    int x;  
  
    public int bar() {  
        int y;  
        y = x;  
        return y;  
    }  
}
```

Does this compile?

What happens at runtime?

Initialized Fields

```
class A {  
    A x;  
  
    public int bar() {  
        return x.bar2();  
    }  
  
    public int bar2() {  
        return 2;  
    }  
}
```

Does this compile?

What happens at runtime?

Initialized Array Components

```
int[] arr;  
int y;  
  
arr = new int[5];  
y = arr[3];
```

Does this compile?

What happens at runtime?

Initialized Array Components

```
int[] arr;
```

```
int y;
```

```
arr = new int[5];
```

```
y = arr[3];
```

Does this compile?

compilation error
(semantic analysis)

Initialized Formal Parameters

```
class A {  
    public int bar(int x) {  
        int y;  
        y = x;  
        return y;  
    }  
}
```

Does this compile?

What happens at runtime?

Initialization in Java

4.12.5. Initial Values of Variables

Every variable in a program must have a value before its value is used:

- Each class variable, instance variable, or array component is initialized with a *default value* when it is created ([§15.9](#), [§15.10](#)):
 - For type `byte`, the default value is zero, that is, the value of `(byte) 0`.
 - For type `short`, the default value is zero, that is, the value of `(short) 0`.
 - For type `int`, the default value is zero, that is, `0`.
 - For type `long`, the default value is zero, that is, `0L`.
 - ...
 - For type `boolean`, the default value is `false`.
 - For all reference types ([§4.3](#)), the default value is `null`.
- Each method parameter ([§8.4.1](#)) is initialized to the corresponding argument value provided by the invoker of the method ([§15.12](#)).

Initialization in Java

- A local variable ([§14.4](#), [§14.14](#)) must be explicitly given a value before it is used, by either initialization ([§14.4](#)) or assignment ([§15.26](#)), in a way that can be verified using the rules for definite assignment ([§16](#)).

Definite Initialization

```
int x;  
int y;  
if (...) {  
    x = 5;  
}  
y = x;
```

Does this compile?

What happens at runtime?

Definite Initialization: Static Analysis

```
int x;  
int y;  
int n;  
n = 5;  
if (n > 2) {  
    x = 5;  
} else {  
}  
y = x;
```

Does this compile?

What happens at runtime?

compilation error
(semantic analysis)

overapproximation

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS
OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

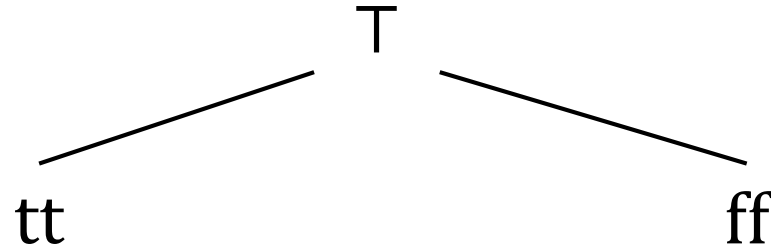
Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

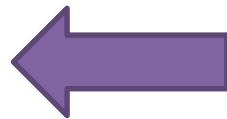


Is-Initialized (Join-Semi)Lattice

$tt \sqsubseteq tt$
 $ff \sqsubseteq ff$
 $tt, ff \sqsubseteq T$
 $T \sqsubseteq T$

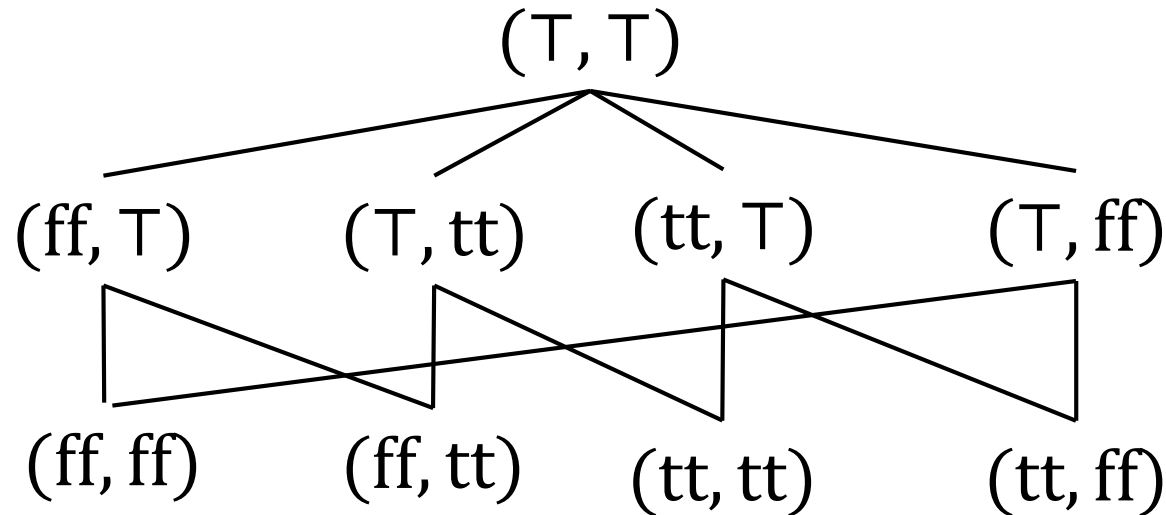


$tt \sqcup tt ?$ tt
 $ff \sqcup ff ?$ ff
 $tt \sqcup ff ?$ T
 $tt \sqcup T ?$ T
 $ff \sqcup T ?$ T



Abstract Domain

- Each element is a map from variables to $\{tt, ff, \top\}$
 $[x_1 \mapsto a_1, x_2 \mapsto a_2, \dots, x_m \mapsto a_m], a_i \in \{tt, ff, \top\}$
- Order: $[x_1 \mapsto a_1, \dots, x_m \mapsto a_m] \sqsubseteq [x_1 \mapsto b_1, \dots, x_m \mapsto b_m]$ iff
 $a_1 \sqsubseteq b_1, \dots, a_m \sqsubseteq b_m$



Abstract Domain

- Each element is a map from variables to $\{\text{tt}, \text{ff}, \top\}$
 $[x_1 \mapsto a_1, x_2 \mapsto a_2, \dots, x_m \mapsto a_m], a_i \in \{\text{tt}, \text{ff}, \top\}$
- Order: $[x_1 \mapsto a_1, \dots, x_m \mapsto a_m] \sqsubseteq [x_1 \mapsto b_1, \dots, x_m \mapsto b_m]$ iff
 $a_1 \sqsubseteq b_1, \dots, a_m \sqsubseteq b_m$
- Join: $[x_1 \mapsto a_1, \dots, x_m \mapsto a_m] \sqcup [x_1 \mapsto b_1, \dots, x_m \mapsto b_m] =$
 $[x_1 \mapsto a_1 \sqcup b_1, \dots, x_m \mapsto a_m \sqcup b_m]$
- Transformers.

Uninitialized Variables

```
int x; [x ↦ ff, y ↦ ff]  
int y; [x ↦ ff, y ↦ ff]  
y = x; x is not initialized
```

Assignment

```
int x; [x ↦ ff, y ↦ ff]
int y; [x ↦ ff, y ↦ ff]
x = 5; [x ↦ tt, y ↦ ff]
y = x;
```

Assignment

```
int x; [x ↦ ff, y ↦ ff]
int y; [x ↦ ff, y ↦ ff]
y = 7; [x ↦ ff, y ↦ ff]
y = x; x is not initialized
```

If

```
int x;      [x ↦ ff, y ↦ ff]
int y;      [x ↦ ff, y ↦ ff]
if (...) {  [x ↦ ff, y ↦ ff]
            [x ↦ ff, y ↦ ff]
            x = 5; [x ↦ tt, y ↦ ff]
} else {    [x ↦ ff, y ↦ ff]
}
           [x ↦ ff, y ↦ ff] ⊔ [x ↦ tt, y ↦ ff]
           = [x ↦ ⊤, y ↦ ff]
```

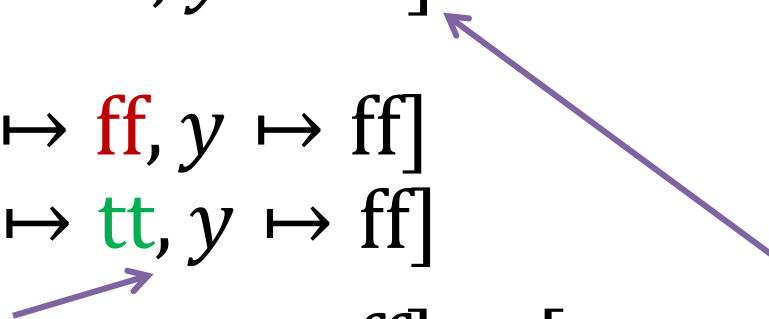
y = x;

compilation error
(semantic analysis)

If

```
int x;      [x ↦ ff, y ↦ ff]
int y;      [x ↦ ff, y ↦ ff]
if (...) {
    x = 5;  [x ↦ ff, y ↦ ff]
           [x ↦ ff, y ↦ ff]
           [x ↦ tt, y ↦ ff]
} else {
    x = 7;  [x ↦ ff, y ↦ ff]
           [x ↦ tt, y ↦ ff]
}
           [x ↦ tt, y ↦ ff] ⊔ [x ↦ tt, y ↦ ff]
           = [x ↦ tt, y ↦ ff]

y = x;
```



Inside a Branch

```
int x;           [x ↦ ff, y ↦ ff]
int y;           [x ↦ ff, y ↦ ff]
if (...) {
    x = 5;       [x ↦ ff, y ↦ ff]
                [x ↦ tt, y ↦ ff]
    y = x;
} else {
}
}
```

While

```
int x;      [x ↦ ff, y ↦ ff]
int y;      [x ↦ ff, y ↦ ff]
while (...) {
    x = 5;  [x ↦ tt, y ↦ ff]
}
           [x ↦ ff, y ↦ ff] ⊔ [x ↦ tt, y ↦ ff]
           = [x ↦ ⊤, y ↦ ff]
```

y = x;

compilation error
(semantic analysis)

Suggested Implementation

- Set of **definitely** initialized local variables after execution of statement at each point in the AST
- Take the join after visiting children
- Store set on stack before visiting a child corresponding to a branch

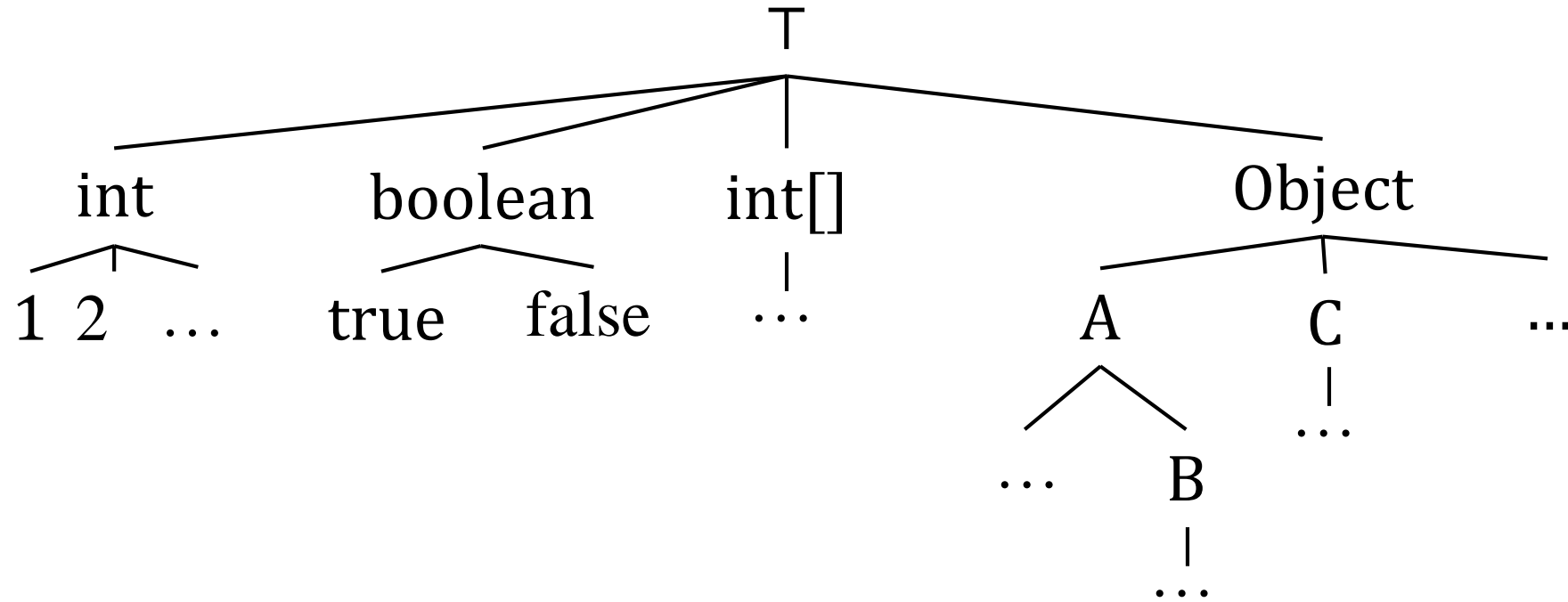
Definite Initialization in Java

- <https://docs.oracle.com/javase/specs/jls/se7/html/jls-16.html>
- Handles if & while conditions a bit more precisely
- Handles all Java features

What If It's a False Alarm?

- In semantic analysis – [part of the spec](#),
part of the interface with the programmer
- Assume the worst and do no harm
 - don't perform the optimization

Static Type Analysis as Static Analysis



Static vs. Dynamic Checks

- Could we prove at **compile time** (= semantic checks, static analysis) that array accesses are in bounds?
 - In some cases, but not all of them
 - Unless we restrict the programmer, and forbid (many) valid programs
- Could we check initialization at runtime instead?
 - Yes, with overhead
 - (Is it worth it?)

Summary

- Initialization in Java
- Definite initialization in Java
- Static analysis
- Abstract interpretation
- Ex. 3