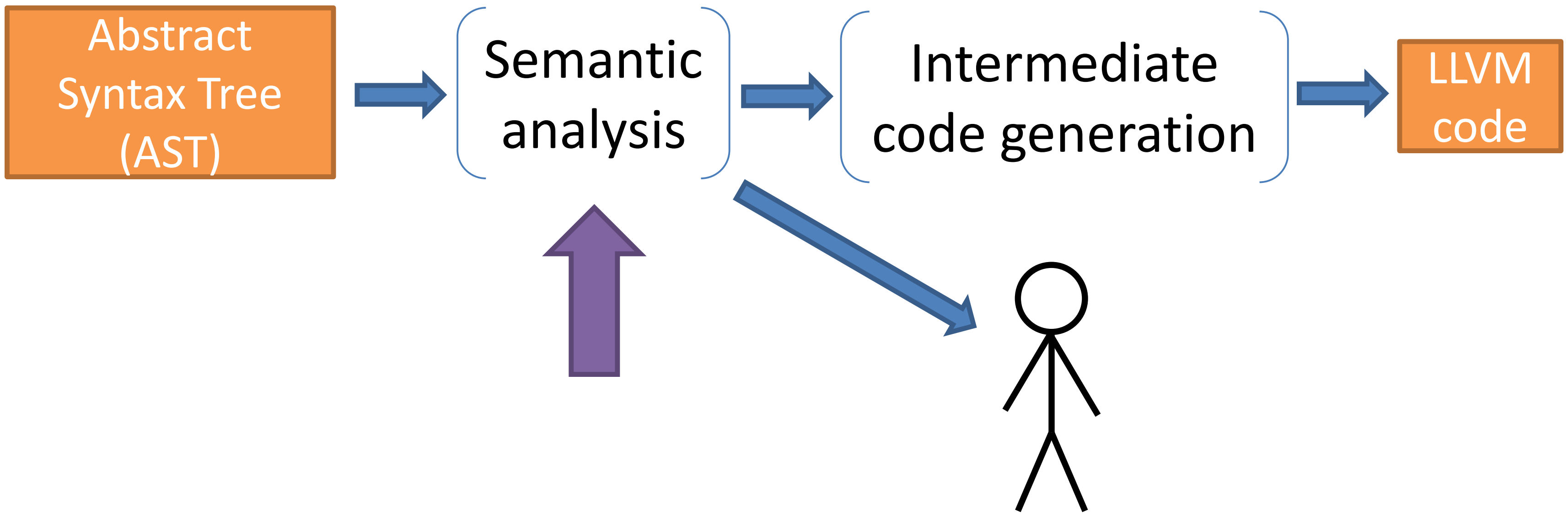
Compiler Construction
Winter 2020

Recitation 7:
Semantic analysis & type checking

Yotam Feldman

Based on slides by Guy Golan-Gueta

Semantic Analysis



Semantic Analysis

Syntactically valid programs may be erroneous

```
int a;
a = "hello";
```

Assigning wrong type

```
int a;
b = 1;
```

undeclared variable as l-value

```
int a;
int a;
a = 1;
```

Variable re-declaration

Scope Rules

- Variable re-declaration
- Reference to undefined name
  - Variable name, method name, class name
- Use the symbol tables to check
  - Re-declaration: symbol already defined at the current scope
  - Undefined: declared somewhere in the hierarchy

Symbol	Kind	Decl	Properties
b	var	int	…

Examples of Type Errors

assigned type doesn't match declared type

```
int a; a = true;
```

binary operator < applied to non-int type

```
1 < true
```

```
void foo(int x) {
    int y;
    foo(5,7);
}
```

argument list doesn't match formal parameters

```
class A {…}
class B extends A {
    void foo() {
      A a;
      B b;
      b = a;
    }
}
```

a is not a subtype of **b**

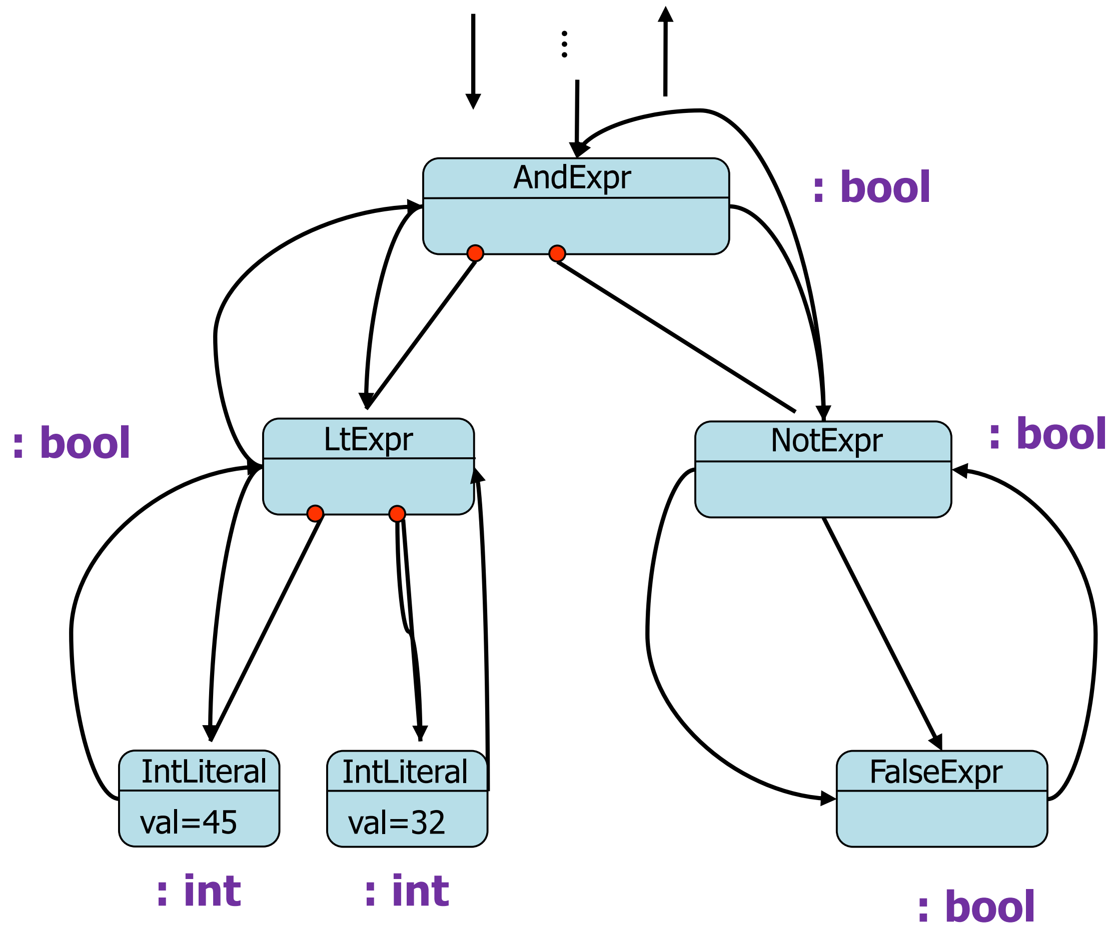
Types

- Type
  - Set of possible values (and operations)
    - **`boolean`** = {**`true,false`**}
    - **`int`** = $\{-2^{31}..2^{31}-1\}$
    - **`void`** = {}
- Type safety: all operations performed on typed values are valid
- Expressions have a type
- Statements use types

Type Analysis and Checking

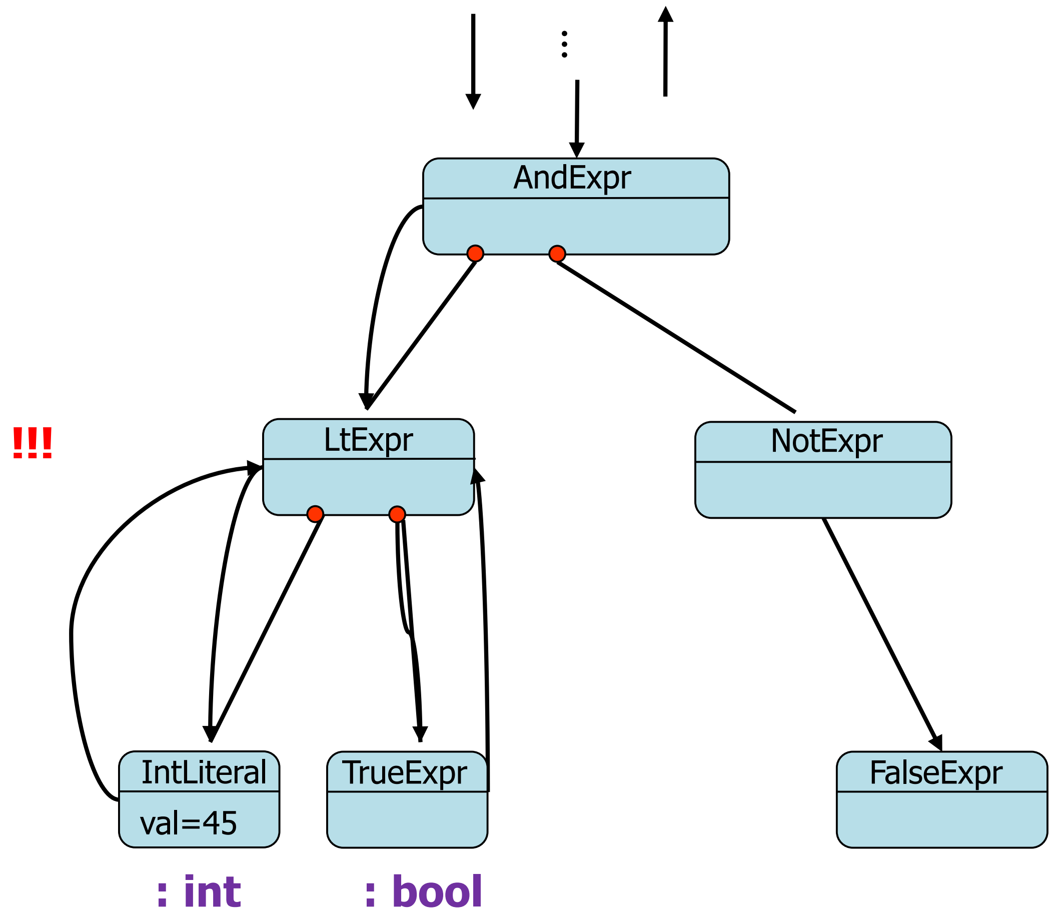
- Compute type for each expression
- Validate type for each expression and in each statement
- Recursively over AST (using visitor)

Type Analysis of Expressions




```
45 < 32 && !false
```

Type Analysis of Expressions

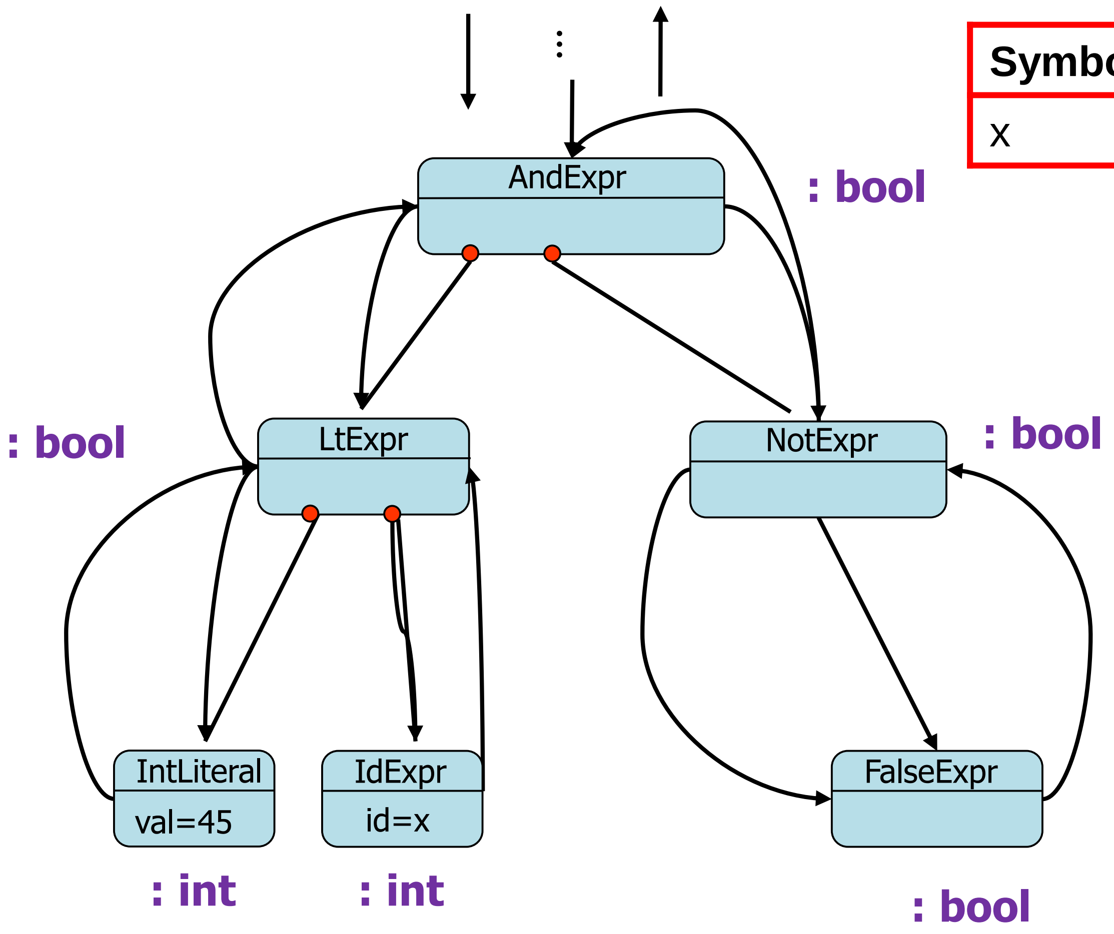


`45 < true && !false`

Type Error

Expressions and Context

Symbol	Kind	Decl	Properties
x	var	int	…




```
45 < x && !false
```

More Expressions in MiniJava

- arr[e] : `int` arr: int[], e: int
- arr.length : `int` arr: int[]
- new int[e] : `int[]` e: int
- new A() : `A`
- this : `current class`
- …
- e.f(e_1,…,e_n) : `t` We’ll get back to it

Type Analysis of Statements in MiniJava

if (e) **then** s_1 **else** s_2	e: boolean
System.out.println(e)	e: int
arr[e_1] = e_2	arr: int[], e_1 : int, e_2 : int
…	
x = e	~~∃t … e: t~~

Assignment and Inheritance

```
class A {
      …
}
class B extends A {
      …
}
```

```
…
A x = new B();
…
```

Subtyping

- Inheritance induces subtyping relation ≤
  - S ≤ T $\Rightarrow$ values(S) $\subseteq$ values(T)
  - "A value of type S may be used wherever a value of type T is expected"

  (Liskov substitution principle?)

Subtyping Examples

1.	int ≤ int ?	yes
2.	int ≤ boolean ?	no
3.	int ≤ A ?	no
4.	A ≤ A ?	yes
5.	A ≤ B ?	if A is a subclass of B (extends it or transitively)

In Java:

1.	null ≤ A ?	yes
2.	null ≤ boolean ?	no
3.	null ≤ boolean[] ?	yes
4.	A[] ≤ B[] ?	yes – in retrospect, might be a mistake

Type Analysis of Statements in MiniJava

if (e) **then** s_1 **else** s_2	e: boolean
System.out.println(e)	e: int
$arr[e_1] = e_2$	arr: int[], e_1 : int, e_2 : int
…	
x = e	$\exists t, t'.\ t' \leq t$, x: t, e: t'
return e	$\exists t'.$ e: t', return type t, $t' \leq t$

More Expressions

- arr[e] : `int` — arr: int[], e: int
- arr.length : `int` — arr: int[]
- new int[e] : `int[]` — e: int
- new A() : `A`
- this : `current context`
- …
- $e.f(e_1,\ldots,e_n)$: `t` — $\exists A.\ e: A,\ A.f(t_1,\ldots,t_n)$ defined with return type t, and $\exists t'_1 \leq t_1, \ldots, t'_n \leq t_n\ .\ e_1 : t'_1, \ldots, e_n : t'_n$

Other Checks: Correct Overriding in MiniJava

```
class A {
      public T_ret bar(T_1 x_1,…,T_n x_n)
      {…}
}
class B extends A {
      public T'_ret bar(T'_1 y_1,…,T'_m y_m)
      {…}
}
```

$m=n$

$T'_1=T_1$, … , $T'_n=T_n$

$T'_{ret} \leq T_{ret}$

Covariant return type

Still Other Checks

- Full list of checks for the exercise
- Ensuring that variables are initialized – next week

Type Analysis and Code Generation

Generating code (or renaming the method) for

```
new B().getField().bar(1, 2);
```

requires the static type of `new B().getField()`

Typically, a type analysis is performed and the type of every expression is stored in the AST

Summary

- Semantic analysis
- Scope checking
- Type checking
- Other semantic checks
- Next week:
  Ensuring that variables are initialized by static analysis