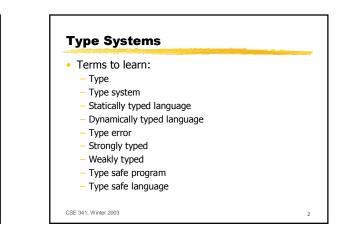
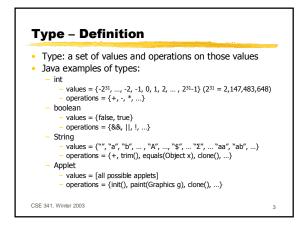
### **Type Systems and Semantics**

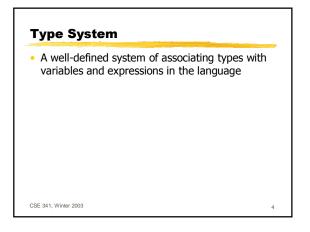
This material covered in Chapter 3 of the text

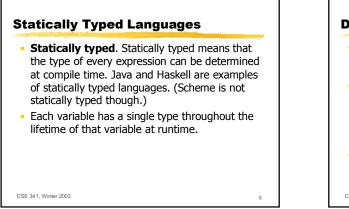
- Syntax versus semantics
- Types
- Formal descriptions of programming language semantics
  - Operational semantics
  - Axiomatic semantics (just skim this section in book)
  - Denotational semantics

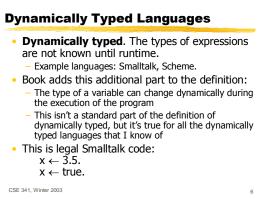
CSE 341, Winter 2003



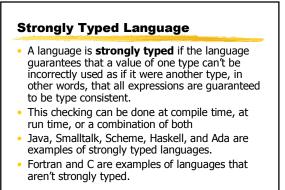




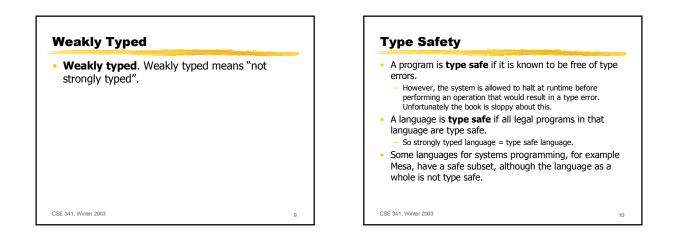




Type error	_
• A type error is a runtime error that occurs when we attempt an operation on a value for which that operation is not defined.	-
• Examples: boolean b, c; b = c+1;	
int i; boolean b; i = b;	
CSE 341, Winter 2003	7



CSE 341, Winter 2003



11

### Tradeoffs

- Generally we want languages to be type safe.
- An exception is a language used for some kinds of systems programming, for example writing a garbage collector. The "safe subset" approach is one way to deal with this problem.
- Advantages of static typing:
  catch errors at compile time
  - machine-checkable documentation
- potential for improved efficiency
- Advantages of dynamic typing:
  - Flexibility
  - rapid prototyping

CSE 341, Winter 2003

## Terminology about Types -Problems

- Unfortunately different authors sometimes use different definitions for the terms "statically typed" and "strongly typed".
- Statically typed. The book defines "statically typed" to mean that the compiler can statically assign a type to every expression – but that type might be wrong.
  By this definition C and Fortran are statically typed.
  - Other authors define "statically typed" to also imply "type safe".
- **Strongly typed**. The book equate strongly typed and type safe (sloppily ...)
- For other authors, strongly typed implies type safe and statically typed. (Is Scheme strongly typed?)
- To avoid misunderstanding, one can describe a language as e.g. "type safe and statically typed".
- CSE 341, Winter 2003

12

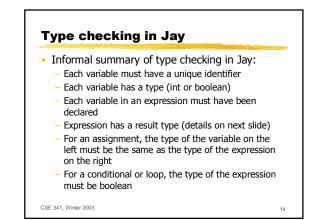
### Jay

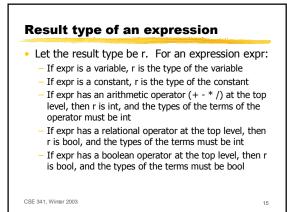
 Jay is a toy language used in the text to illustrate language concepts. In its original form it has no procedures or functions, and no userdefined types

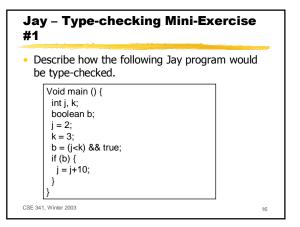
13

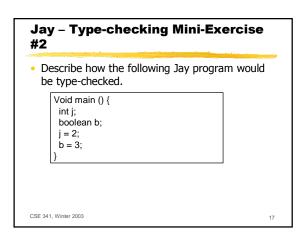
Jay is statically and strongly typed

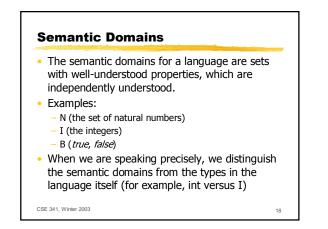
CSE 341, Winter 2003

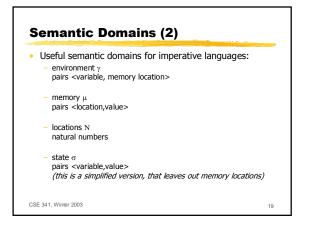


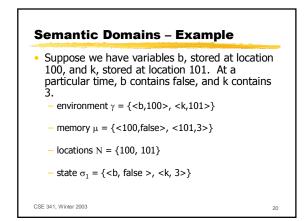


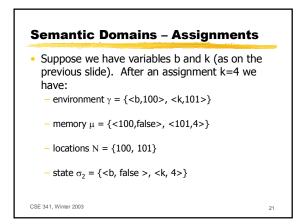


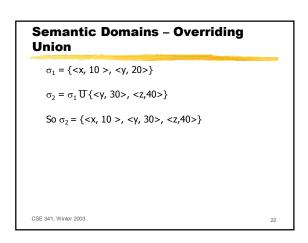


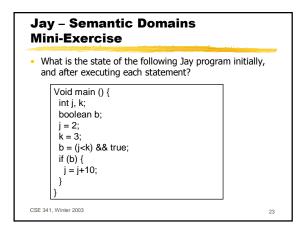


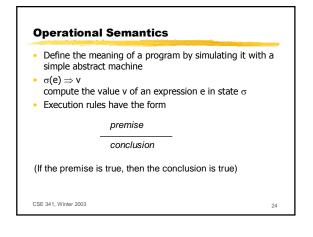




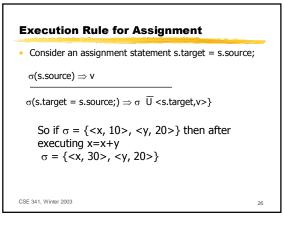


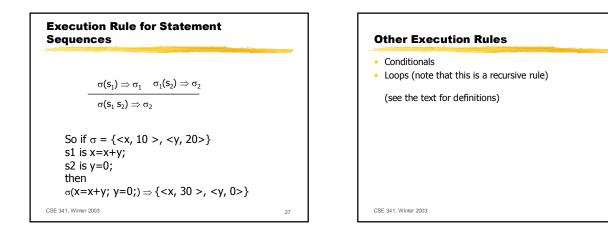


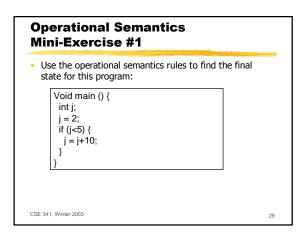


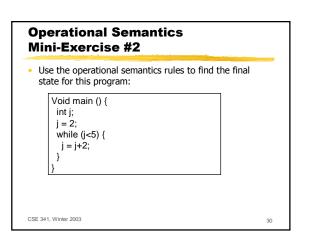


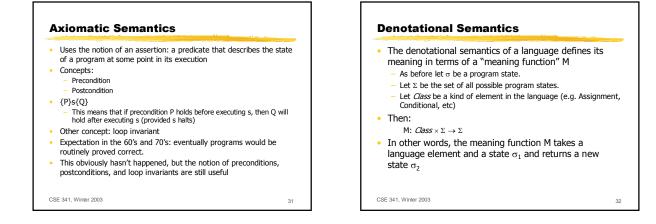
Execution Rules - Examples	
Execution rule for addition:	
$\sigma(e_1) \Rightarrow v_1  \sigma(e_2) \Rightarrow v_2$	
$\sigma(\mathbf{e}_1 + \mathbf{e}_2) \Rightarrow \mathbf{v}_1 + \mathbf{v}_2$	
So if $\sigma = \{, \}$ then $\sigma(x + y) \Rightarrow 30$	
CSE 341, Winter 2003	25

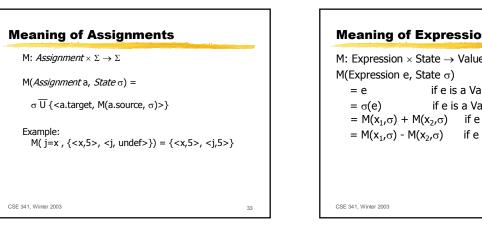












# **More on Denotational Semantics** (Optional, Extra Material)

In most papers on denotational semantics, the meaning function M is written using double brackets: [[e]] σ rather than M(e,σ)

- Strictly speaking, the meaning function applied to a constant takes an element of the language into a semantic domain - these two domains are different:  $[[12]] \, \sigma = \mathit{12}$
- There is a copy of a tutorial on denotational semantics by R.D. Tennent linked from the 341 website for those who would like to learn more.

```
CSE 341, Winter 2003
```

**Meaning of Expressions** M: Expression  $\times$  State  $\rightarrow$  Value if e is a Value if e is a Variable  $= M(x_1, \sigma) + M(x_2, \sigma)$  if  $e = x_1 + x_2$ if  $e = x_1 - x_2$ 34

