6.035

Spring 2010

Semantic Analysis

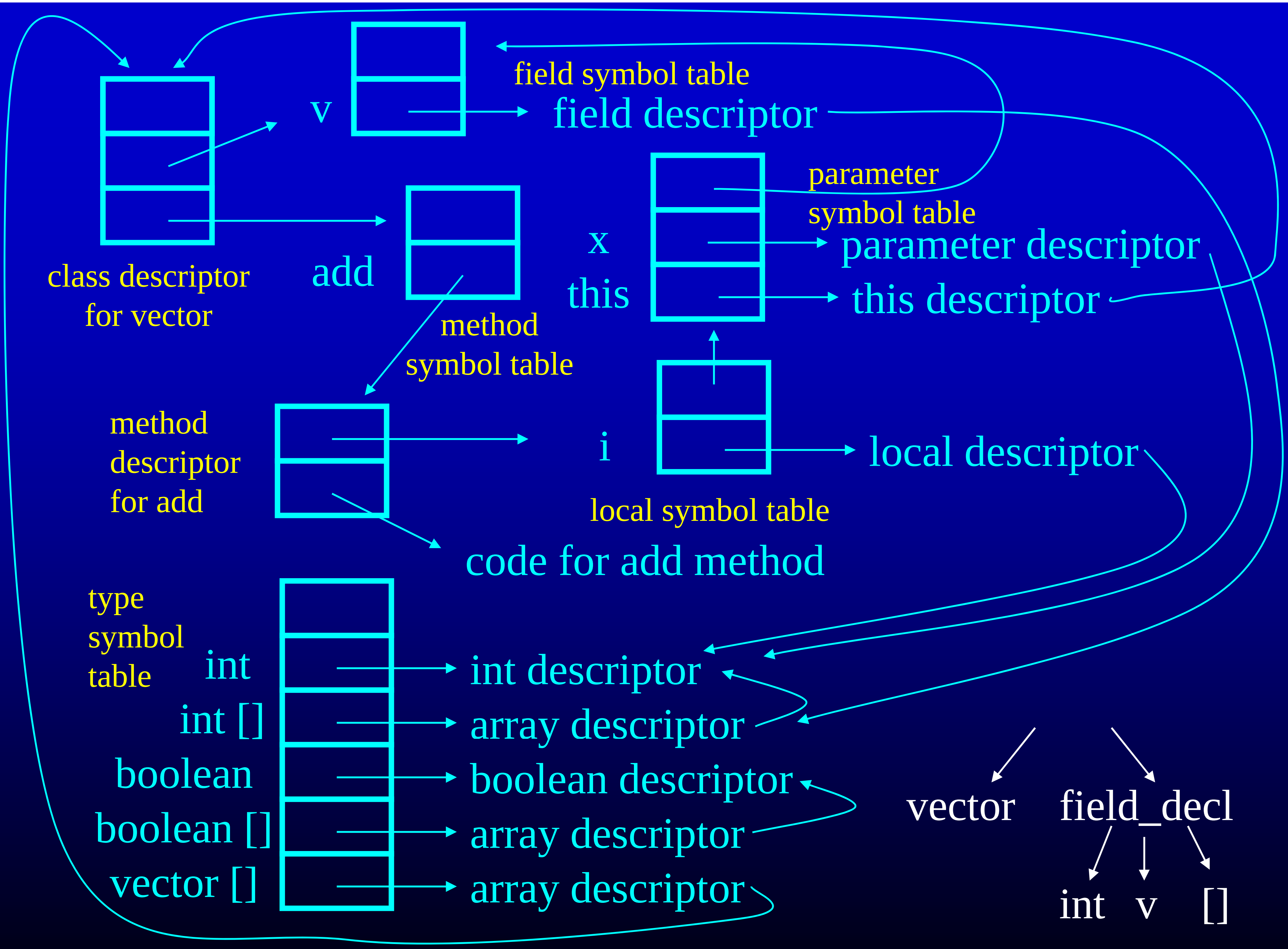
Saman Amarasinghe

Massachusetts Institute of Technology

Symbol Table Summary

- Program Symbol Table (Class Descriptors)
- Class Descriptors
  - Field Symbol Table (Field Descriptors)
    - Pointer to Field Symbol Table for SuperClass
  - Method Symbol Table (Method Descriptors)
    - Pointer to Method Symbol Table for Superclass
- Method Descriptors
  - Local Variable Symbol Table (Local Variable Descriptors)
    - Parameter Symbol Table (Parameter Descriptors)
      - Pointer to Field Symbol Table of Receiver Class

- Local, Parameter and Field Descriptors
  - Type Descriptors in Type Symbol Table or Class Descriptors


class descriptor for vector
v
field symbol table
field descriptor
add
method symbol table
x
this
parameter symbol table
parameter descriptor
this descriptor
method descriptor for add
i
local descriptor
local symbol table
code for add method
type symbol table
int
int []
boolean
boolean []
vector []
int descriptor
array descriptor
boolean descriptor
array descriptor
array descriptor
vector
field_decl
int
v
[]

Outline

- Practical Issues in Intermediate Representation
- What is semantic analysis?
- Type systems
- What to check?

How to Store Statements

x = a*b + c

- Flat Lists
  - Need to represent intermediate values
    - In a stack

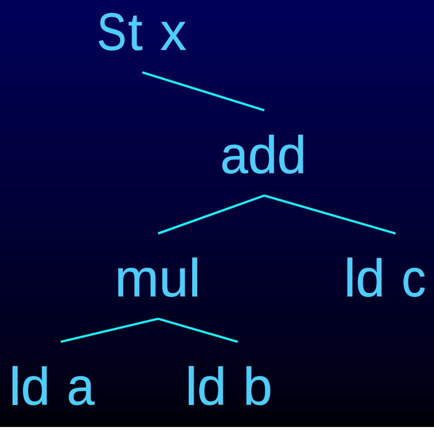
      push a; push b; mul; push c; add; pop x

    - In single use temporary registers

      t1 = mul a, b

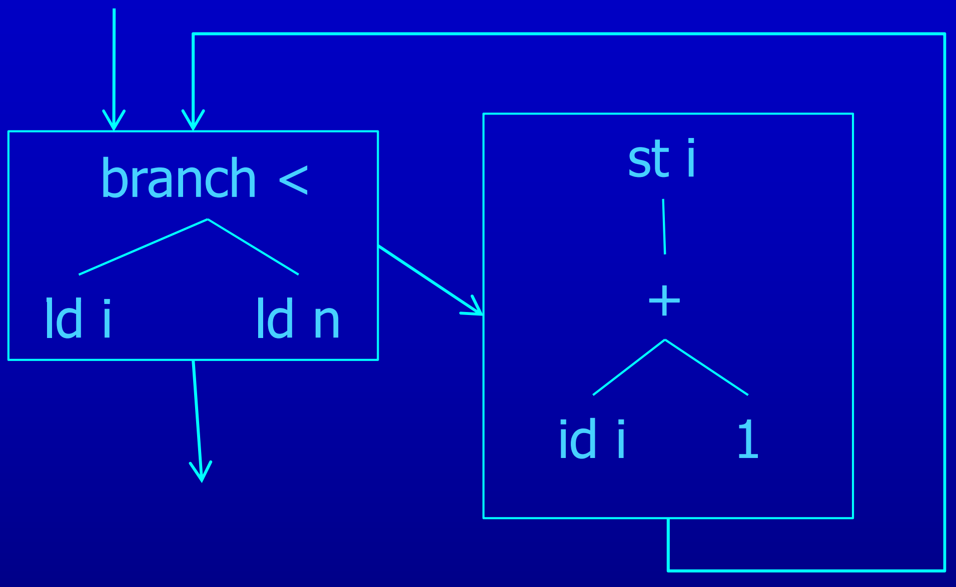
      x − add t1, c

- Trees
  - Intermediate values are implicit in the edges

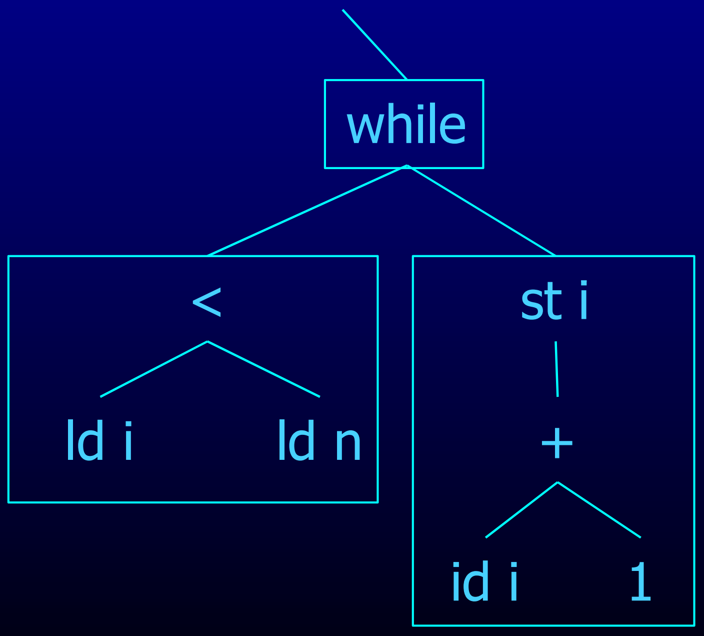


Handling Control-Flow

- Control-Flow Graph
  - Pros: Simple, uniform
  - Cons: lost the high level structure



- Structured Control Flow Graph
  - Pros: Help in loop optimizations and parallelization
  - Cons: Many different types of nodes

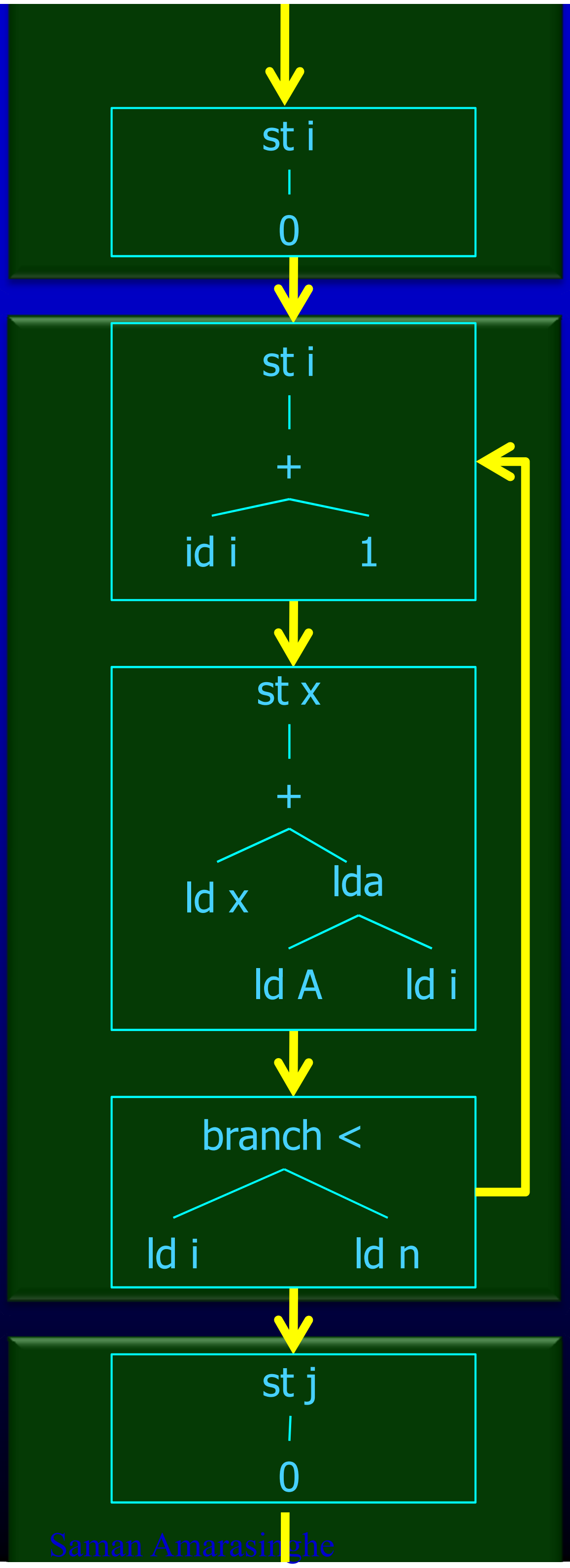


Basic Blocks

- Group statements into larger chunks
  - Helps in the optimization phase

- Basic Block
  - Single entry point at top
  - Linear collection of statements
  - No control transfer instructions in the middle
  - Only last instruction can be a control transfer

Basic Blocks



What not to do!

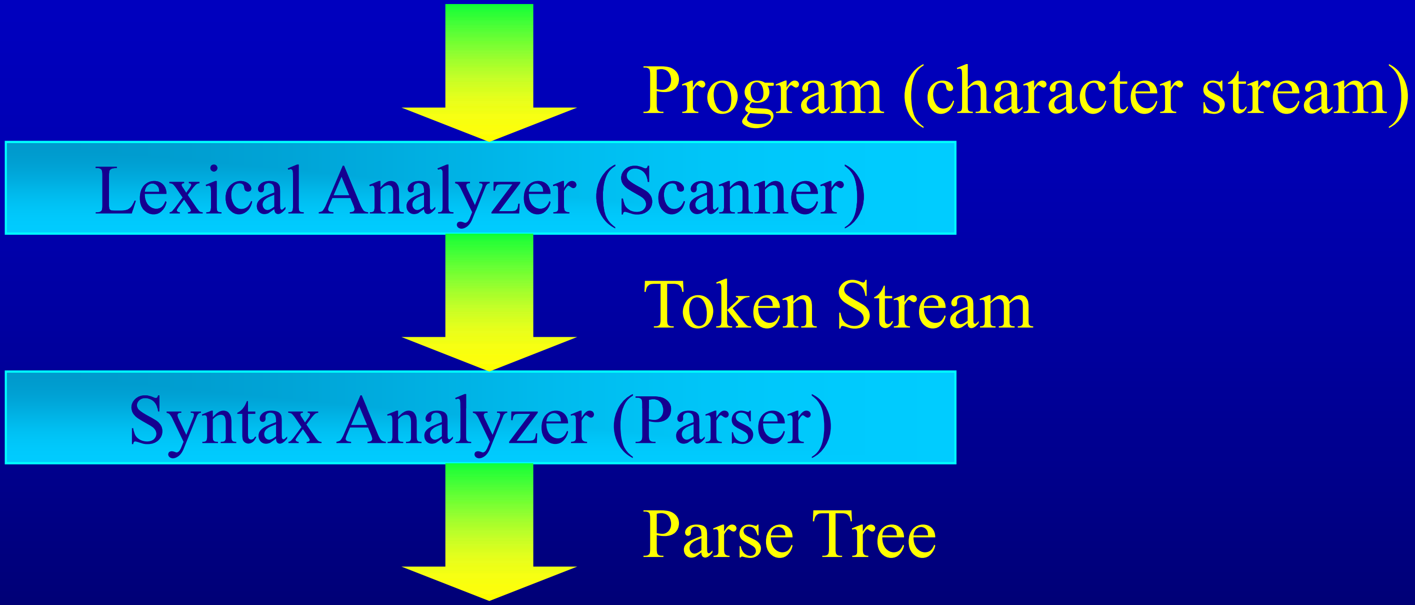
- Keep data in the abstract (in descriptors)
  - Don't try to do register allocation!
- No optimizations!
  - Even when they seem sooo easy

- Theme:
  - take small steps
  - don't try to do too many at once
  - don't try to do anything too early
  - try not to loose any information!

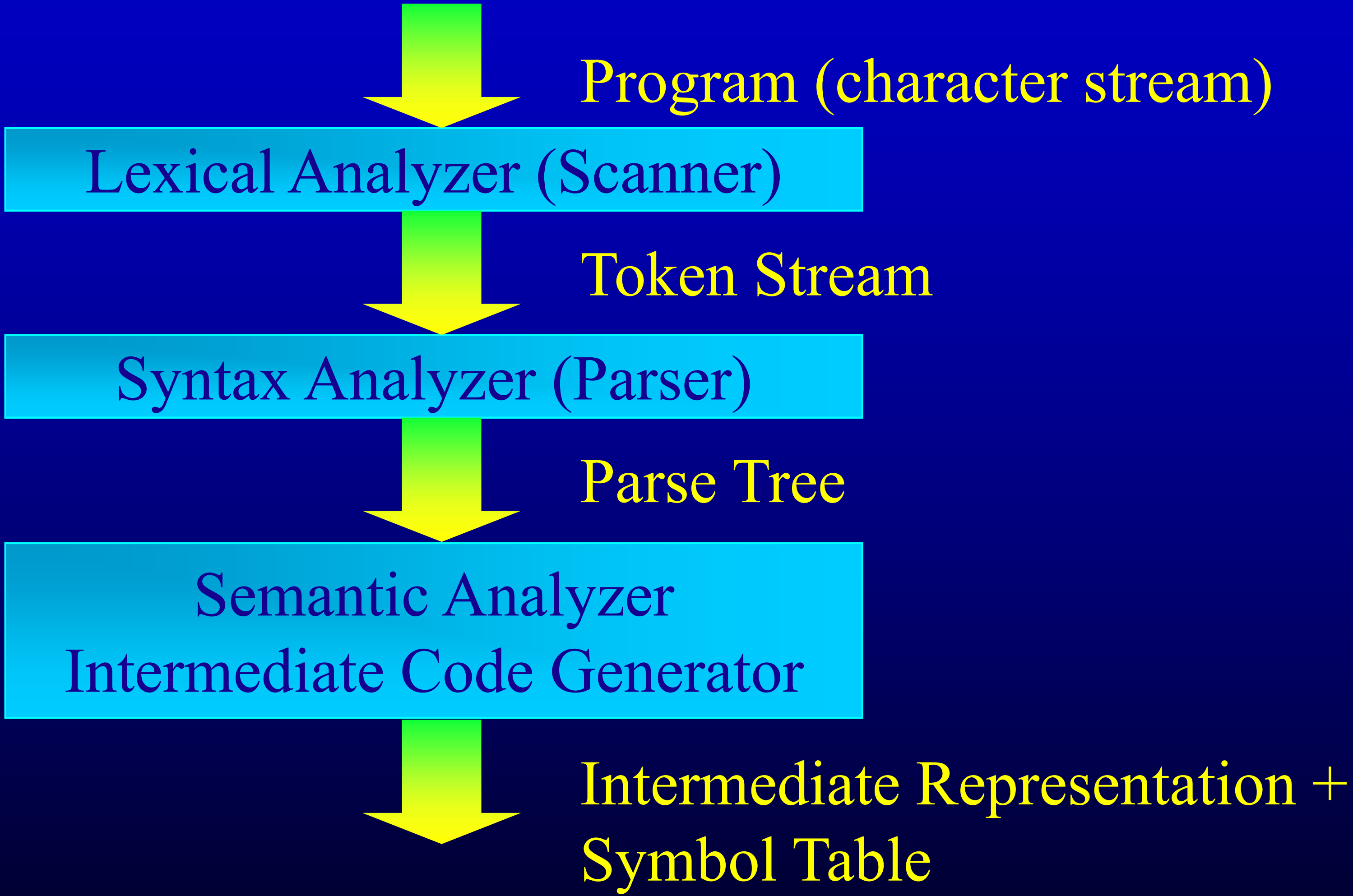
Outline

- Practical Issues in Intermediate Representation
- **What is semantic analysis?**
- Type systems
- What to check?

Where are we?


Program (character stream)
Lexical Analyzer (Scanner)
Token Stream
Syntax Analyzer (Parser)
Parse Tree

Where are we?


Program (character stream)
Lexical Analyzer (Scanner)
Token Stream
Syntax Analyzer (Parser)
Parse Tree
Semantic Analyzer
Intermediate Code Generator
Intermediate Representation +
Symbol Table

What is the semantics of a program?

- Syntax
  - How a program looks like
  - Textual representation or structure
  - A precise mathematical definition is possible
- Semantics
  - What is the meaning of a program
  - Harder to give a mathematical definition

Why do semantic checking?

- Make sure the program confirms to the programming language definition
- Provide meaningful error messages to the user
- Don’t need to do additional work, will discover in the process of intermediate representation generation

Semantic Checking

- Static checks vs. Dynamic checks
- Static checks
  - Flow-of-control checks
  - Uniqueness checks
  - Type checks

Flow of control checks

- Flow-control of the program is context sensitive
- Examples:
  - Declaration of a variable should be visible at use (in scope)
  - Declaration of a variable should be before use
  - Each exit path returns a value of the correct type
- What else?

Uniqueness checks

- Use and misuse of identifiers
  - Cannot represent in a CFG (same token)
- Examples:
  - No identifier can be used for two different definitions in the same scope

Type checks

- Most extensive semantic checks
- Examples:
  - Number of arguments matches the number of formals and the corresponding types are equivalent
  - If called as an expression, should return a type
  - Each access of a variable should match the declaration (arrays, structures etc.)
  - Identifiers in an expression should be “evaluatable”
  - LHS of an assignment should be “assignable”
  - In an expression all the types of variables, method return types and operators should be “compatible”

Dynamic checks

- Array bounds check
- Null pointer dereference check

Outline

- Practical Issues in Intermediate Representation
- What is semantic analysis?
- **Type systems**
- What to check?

Type Systems

- A type system is used to for the type checking
- A type system incorporates
  - syntactic constructs of the language
  - notion of types
  - rules for assigning types to language constructs

Type expressions

- A compound type is denoted by a type expression
- A type expression is
  - a basic type
  - application of a type constructor to other type expressions

Type Expressions: Basic types

- Atomic types defined by the language
- Examples:
  - integers
  - booleans
  - floats
  - characters
- type_error
  - special type that'll signal an error
- void
  - basic type denoting "the absence of a value"

Type Expressions: Names

- Since type expressions maybe be named, a type name is a type expression

Type Expressions: Products

- If T_1 and T_2 are type expressions $T_1 \times T_2$ is also a type expression

Type Expressions: Arrays

- If T is a type expression an **array(T, I)** is also a type expression
  - I is a integer constant denoting the number of elements of type T
  - Example:

```
int foo[128];
```

    array(integer, 128)

Type Expressions: Method Calls

- Mathematically a function maps
  - elements of one set (the domain)
  - to elements of another set (the range)
- Example

```
int foobar(int a, boolean b, int c)
```

integer $\times$ boolean $\times$ integer $\rightarrow$ integer

Type Expressions: Some others

- Records
  - structures and classes
  - Example

    ```
    class { int i; int j;}
    ```

    integer $\times$ integer
- Functional Languages
  - functions that take functions and return functions
  - Example

    (integer $\rightarrow$ integer) $\times$ integer $\rightarrow$ (integer $\rightarrow$ integer)

A simple typed language

- A language that has a sequence of declarations followed by a single expression

  P $\rightarrow$ D; E

  D $\rightarrow$ D; D | **id** : T

  T $\rightarrow$ **char** | **integer** | **array** [**num**] **of** T

  E $\rightarrow$ **literal** | **num** | **id** | E + E | E [E]

- Example Program

```
var: integer;
var + 1023
```

A simple typed language

- A language that has a sequence of declarations followed by a single expression

  P $\rightarrow$ D; E

  D $\rightarrow$ D; D | **id** : T

  T $\rightarrow$ **char** | **integer** | **array** [**num**] **of** T

  E $\rightarrow$ **literal** | **num** | **id** | E + E | E [E]

- What are the semantic rules of this language?

Parser actions

P $\rightarrow$ D; E

D $\rightarrow$ D; D

D $\rightarrow$ **id** : T { addtype(id.entry, T.type); }

T $\rightarrow$ char { T.type = char; }

T $\rightarrow$ integer { T.type = integer; }

T $\rightarrow$ array [num] of T_1

{ T.type = array(T_1.type, num.val); }

Parser actions

E $\rightarrow$ literal { E.type − char; }

E $\rightarrow$ num { E.type = integer; }

E $\rightarrow$ id { E.type = lookup_type(id.name); }

Parser actions

```
E → E1 + E2        { if $E_1$.type == integer and
                         $E_2$ .type == integer  then
                             E.type = integer
                     else
                             E.type = type_error
                   }
```

Parser actions

```
$E \rightarrow E_1\ [E_2\ ]$        { if $E_2$.type – – integer and
                                  $E_1$ .type == array(s, t)  then
                                      E.type – s
                              else
                                      E.type = type_error
                          }
```

Type Equivalence

- How do we know if two types are equal?
  - Same type entry
  - Example:

```
int A[128];
foo(A);

foo(int B[128]) { … }
```

  - Two different type entries in different symbol tables
  - But they should be the same

Structural Equivalence

- If the type expression of two types have the same construction, then they are equivalent
- "Same construction"
  - Equivalent base types
  - Same set of type constructors are applied in the same order (i.e. equivalent type tree)

Type Coercion

- Implicit conversion of one type to another type
- Example

```
int A;
float B;
B = B + A
```

- Two types of coercion
  - widening conversions
  - narrowing conversions

Narrowing conversions

- Conversions that may loose information
- Examples:
  - integers to chars
  - longs to shorts
- Rare in languages

Widening conversions

- Conversions without loss of information
- Examples:
  - integers to floats
  - shorts to longs
- What is done in many languages (including decaf)

Widening Conversions

- Basic Principle: Hierarchy of number types
  - int → float → double
- All coercions go up hierarchy
  - int to float;
  - int, float to double
- Result is type of operand highest up in hierarchy
  - int + float is float
  - int + double is double
  - float + double is double

Type casting

- Explicit conversion from one type to another
- Both widening and narrowing
- Example

```
int A;
float B;
A − A + (int)B
```

- Unlimited typecasting can be dangerous

Question:

- Can we assign a single type to all variables, functions and operators?
- How about +, what is its type?

Overloading

- Some operators may have more than one type.
- Example

```
int A, B, C;
float X, Y, Z;
A = A + B
X = X + Y
```

- Complicates the type system
  - Example

    ```
    A = A + X
    ```

    - What is the type of + ?

Outline

- Practical Issues in Intermediate Representation
- What is semantic analysis?
- Type systems
- **What to check?**

Parameter Descriptors

- When build parameter descriptor, have
  - name of type
  - name of parameter
- What is the check?
  - Is name of type identifies a valid type?
    - look up name in type symbol table
    - if not there, look up name in program symbol table (might be a class type)
    - if not there, fails semantic check

Local Descriptors

- When build local descriptor, have
  - name of type
  - name of local
- What is the check?
  - Is name of type identifies a valid type?
    - look up name in type symbol table
    - if not there, look up name in program symbol table (might be a class type)
    - if not there, fails semantic check

Local Symbol Table

- When building the local symbol table, have a list of local descriptors
- What to check for?
  - duplicate variable names
  - shadowed variable names
- When to check?
  - when insert descriptor into local symbol table
- Parameter and field symbol tables similar

Class Descriptor

- When build class descriptor, have
  - class name and name of superclass
  - field symbol table
  - method symbol table
- What to check?
  - Superclass name corresponds to actual class
  - No name clashes between field names of subclass and superclasses
  - Overridden methods match parameters and return type declarations of superclass

Load Instruction

- What does compiler have? Variable name.
- What does it do? Look up variable name.
  - If in local symbol table, reference local descriptor
  - If in parameter symbol table, reference parameter descriptor
  - If in field symbol table, reference field descriptor
  - If not found, semantic error

Load Array Instruction

- What does compiler have?
  - Variable name
  - Array index expression
- What does compiler do?
  - Look up variable name (if not there, semantic error)
  - Check type of expression (if not integer, semantic error)

Load Array Instruction

What else can/should be checked?

Add Operations

- What does compiler have?
  - two expressions
- What can go wrong?
  - expressions have wrong type
  - must both be integers (for example)
- So compiler checks type of expressions
  - load instructions record type of accessed variable
  - operations record type of produced expression
  - so just check types, if wrong, semantic error

Type Inference for Add Operations

- Most languages let you add floats, ints, doubles
- What are issues?
  - Types of result of add operation
  - Coercions on operands of add operation
- Standard rules usually apply
  - If add an int and a float, coerce the int to a float, do the add with the floats, and the result is a float.
  - If add a float and a double, coerce the float to a double, do the add with the doubles, result is double

Store Instruction

- What does compiler have?
  - Variable name
  - Expression

- What does it do?
  - Look up variable name.
    - If in local symbol table, reference local descriptor
    - If in parameter symbol table, error
    - If in field symbol table, reference field descriptor
    - If not found, semantic error
  - Check type of variable name against type of expression
    - If variable type not compatible with expression type, error

Store Array Instruction

- What does compiler have?
  - Variable name, array index expression
  - Expression
- What does it do?
  - Look up variable name.
    - If in local symbol table, reference local descriptor
    - If in parameter symbol table, error
    - If in field symbol table, reference field descriptor
    - If not found, semantic error

    Check that type of array index expression is integer
  - Check type of variable name against type of expression
    - If variable element type not compatible with expression type, error

Method Invocations

- What does compiler have?
  - method name, receiver expression, actual parameters
- Checks:
  - receiver expression is class type
  - method name is defined in receiver's class type
  - types of actual parameters match types of formal parameters
  - What does match mean?
    - same type?
    - compatible type?

Return Instructions

- What does compiler have?
  - Expression
- Checks:
  - If the return type matches the expression?

Conditional Instructions

- What does compiler have?
  - Expression for the if-condition and the statement list of then (and else) blocks
- Checks:
  - If the conditional expression producing a Boolean value?

Semantic Check Summary

- Do semantic checks when build IR
- Many correspond to making sure entities are there to build correct IR
- Others correspond to simple sanity checks
- Each language has a list that must be checked
- Can flag many potential errors at compile time