

**Winter 2006-2007
Compiler Construction
T13 – Recap**

Mooly Sagiv and Roman Manevich
School of Computer Science
Tel-Aviv University

Today

<small>ic</small>	<small>Lexical Analysis</small>	<small>Syntax Analysis Parsing</small>	<small>AST</small>	<small>Symbol Table etc.</small>	<small>Inter. Rep. (IR)</small>	<small>Code Generation</small>	<small>exe</small>
IC Language							Executable code

- PA4 – have one more week
- Special recitation before exam
 - Send me questions
 - Suggest date on forum
- Register allocation optimization
 - Details on [web site](#)
- PA5
- Recap

2

PA5

- Generate assembly code from LIR
- Suggestion:
 - Set up a working environment for assembling/linking/running
 - Use an example .s file from the web-page
 - Start by translating IC programs with just a main method (ignore function calls) and no objects
 - Add static function calls
 - Add objects and virtual function calls
 - Optional: add optimizations

3

Register allocation constraints

- esi,edi can be also used for (two-operand) arithmetic instructions (not just eax,ebx,ecx,edx)
- idiv – expects input in edx:eax (divisor can be any register) and stores output in edx:eax
- Some unary instructions expect specific registers
- Read manual to find specific details

4

Final exam – 21/2/2007

- Materials taught in class and recitations
- Example exams on web-site
- Popular types of questions
 - Introducing new features to language (IC)
 - Most reasonable Java features good candidates:
 - Access control,
 - Exceptions,
 - Static fields
 - Parsing related questions
 - Building LR parser,
 - Resolving conflicts,
 - Running parser on input
 - Activation records
 - Running small program

5

Example program

```
// An example program
class Hello {
    boolean state;
    static void main(string[] args) {
        Hello h = new Hello();
        boolean s = h.rise();
        Library.println(s);
        h.setState(false);
    }
    boolean rise() {
        boolean oldState = state;
        state = true;
        return oldState;
    }
    void setState(boolean newState) {
        state = newState;
    }
}
```

6

Scanning

```
// An example program
class Hello {
  boolean state;
  static void main(String[] args) {
    Hello h = new Hello();
    boolean s = h.rise();
    Library.println(s);
    h.setState(false);
  }
  boolean rise() {
    boolean oldState = state;
    state = true;
    return oldState;
  }
  void setState(boolean newState) {
    state = newState;
  }
}
```

- Issues in lexical analysis:
- Pattern matching conventions (longest match, priorities)
 - Running scanner automaton
 - Language changes:
 - New keywords,
 - New operators,
 - New meta-language features, e.g., annotations

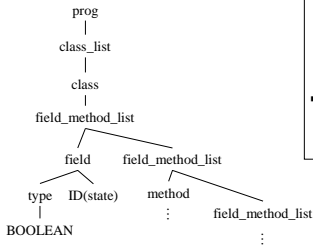
↓ scanner read text and generate token stream

CLASS, CLASS_ID(Hello), LB, BOOLEAN, ID(state), SEMI ...

Parsing and AST

CLASS, CLASS_ID(Hello), LB, BOOLEAN, ID(state), SEMI ...

↓ parser uses stream of token and generate derivation tree

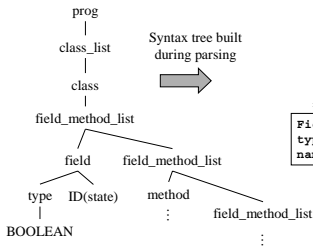


- Issues in syntax analysis:
- Grammars: LL(0), LR(0)
 - Building LR(0) parsers
 - Transition diagram
 - Parse table
 - Running automaton
 - Conflict resolution
 - Factoring
 - In parse table

Parsing and AST

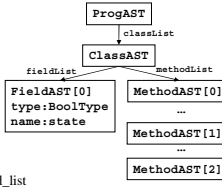
CLASS, CLASS_ID(Hello), LB, BOOLEAN, ID(state), SEMI ...

↓ parser uses stream of token and generate derivation tree



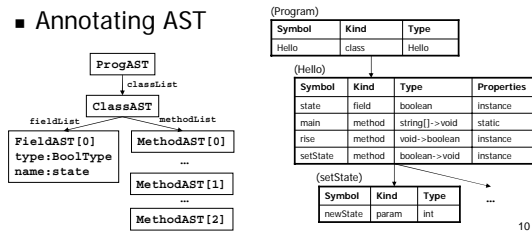
Syntax tree built during parsing

- Should know difference between derivation tree and AST
- Know how to build AST from input



Semantic analysis

- Representing scopes
- Type-checking
- Semantic checks
- Annotating AST



Semantic analysis

- The roles of scopes
 - Provide unique symbol for each identifier – use symbols in next phases instead of identifiers
 - Disambiguate identifiers
 - Determine scope rules: undeclared ids, double-declaration, declaration in wrong scope...
- Type-checking
 - Associate types with identifiers
 - Infer types for expressions
 - Check well-formed statements/classes etc.
 - Know type rule notations

11

Semantic conditions

- What is checked in compile-time and what is checked in runtime?

Event	C/R
Program execution halts	
All execution paths in function include a return statement	
Array index within bound	
In Java the cast statement (Integer) f is legal	
In Java method o.m(...) is illegal since m is private	

12

Semantic conditions

- What is checked in compile-time and what is checked in runtime?

Event	C/R
Program execution halts	R (undecidable in general)
All execution paths in function include a return statement	C (number of simple paths from function entry to exit is finite)
Array index within bound	R (undecidable in general)
In Java the cast statement (A) f is legal	Depends: if A is sub-type of f then checked during runtime (raising exception), otherwise flagged as an error during compilation
In Java method o.m(...) is illegal since m is private	C

13

More semantic conditions

```

class A {...}
class B extends A {
  void foo() {
    B[] bArray = new B[10];
    A[] aArray = bArray;
    A x = new A();
    if (...) x = new B();
    aArray[5]=x;
  }
}
    
```

- Explain why the assignment aArray=bArray is considered well-typed in Java.
- Under what conditions should could the assignment aArray[5]=x lead to a runtime error? Explain.
- How does Java handle the problem with the assignment aArray[5]=x?

14

Translation to IR

- Accept annotated AST and translate functions into lists of instructions
 - Compute offsets for fields and virtual functions
- Issues: dispatch tables, weighted register allocation
- Support extensions, e.g., translate switch statements
- Question: give the method tables for **Rectangle** and **Square**

```

class Shape {
  boolean isShape() {return true;}
  boolean isRectangle() {return false;}
  boolean isSquare() {return false;}
  double surfaceArea() {...}
}
class Rectangle extends Shape {
  double surfaceArea() {...}
  boolean isRectangle() {return true;}
}
class Square extends Rectangle {
  boolean isSquare() {return true;}
}
    
```

15

Answer

Method table for rectangle

Shape_isShape
Rectangle_isRectangle
Shape_isSqaure
Rectangle_surfaceArea

Method table for square

Shape_isShape
Rectangle_isRectangle
Sqaure_isSqaure
Rectangle_surfaceArea

16

LIR translation

```
// An example program
class Hello {
  boolean state;
  static void main(string[] args) {
    Hello h = new Hello();
    boolean s = h.rise();
    Library.println(s);
    h.setState(false);
  }
  boolean rise() {
    boolean oldState = state;
    state = true;
    return oldState;
  }
  void setState(boolean newState) {
    state = newState;
  }
}

```

Compute method and field offsets

fieldToOffset	methodToOffset
DVPtr = 0	_Hello_rise = 0
state = 1	_Hello_setState = 1

Sometimes real offsets computed only in code generation

```

_DV_Hello: [Hello_rise,Hello_setState]
_Hello_rise:
  Move this,R0
  MoveField R0.0,R0
  Move R0,oldState
  Return oldState
_Hello_setState:
  Move this,R0
  Move newState,R1
  MoveField R1,newR0.0
_ic_main:
  __allocateObject(8),R0
  MoveField _DV_Hello,R0.0
  Move R0,h
  Move h,R0
  VirtualCall R0.0(),R0
  Move R0,s
  Library.println(s),Rdummy
  Move h,R0
  VirtualCall R0.i(newState=0)

```

17

Possible question

- Suppose we wish to provide type information during runtime, e.g., to support operators like `instanceof` in Java
- Describe the changes in runtime organization needed to support this operator and the translation to IR

18

Answer

- As a very restricted solution, we can avoid any changes to the runtime organization by using the pointers to the dispatch table as the type indicators
- We would translate `x instanceof A` as

```
Move x,R0
MoveField R0.0,R0
Compare R0,_DV_A
```
- The comparison is true iff the dispatch table of the object pointed-to by `x` is the dispatch table of class `A`, meaning that the object is of type `A`

19

Code generation

- Translate IR code to assembly
- Issues:
 - Activation records and call sequences
 - Run simple example and draw frame stacks in different point of execution

20

More questions

21

Possible question

- Support Java override annotation
 - // @Override
 - Annotation is written above method to indicate it overrides a method in superclass
- Describe the phases in the compiler affected by the change and the changes themselves

Legal program

```
class A {  
    void rise() {...}  
}  
class B extends A {  
    // @Override  
    void rise() {...}  
}
```

Illegal program

```
class A {  
    void rise() {...}  
}  
class B extends A {  
    // @Override  
    void ris() {...}  
}
```

22

Answer

- The change affects the lexical analysis, syntax analysis and semantic analysis
- It does not effect later phases since the annotation is meant to add a semantic condition by the user

23

Changes to scanner

- Add pattern for @Override inside comment state patterns
- Add Java code to action to comments – instead of not returning any token, we now return a token for the annotation
- What if we want to support multiple annotations in comments?

```
boolean override=false;  
%%  
<INITIAL> // { override=false; yybegin(comment); }  
<comment> @Override { override=true; }  
<comment> \n { if (override)  
                return new Token(...,override,...)  
            }  
}
```

24

Changes to parser and AST

- Suppose we have rule
 - method \rightarrow static type name params '{ mbody }'
| type name params '{ mbody }'
 - Since that annotation is legal only for instance methods we rewrite the rule into
method \rightarrow static type name params '{ mbody }'
| type name params '{ body }'
| OVERRIDE type name params '{ mbody }'
- We need to add a Boolean flag to the method AST node to indicate that the method is annotated

25

Changes to semantic analysis

- Suppose we have an override annotation above a method m in class A
- We use the following information
 - Symbol tables
 - Class hierarchy
 - Type table (for the types of methods)
- We check the following semantic condition using the following inform
 1. We check that the class A extends a superclass (otherwise it does not make sense to override a method)
 2. We check the superclasses of A by going up the class hierarchy until we find the first method m and check that it has the same signature as $A.m$. If we fail to find such a method we report an error

26

**Good luck
in the exams!**

27
