COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia

`http://www.cse.unsw.edu.au/~cs3131`

`http://www.cse.unsw.edu.au/~cs9102`

Week 10 (Mon): Java Class File Format & Runtime Areas

1. Class file format (Sun's JVM Spec Book)
2. Runtime areas (JVM Spec & Bill Venners' Inside the JVM book)
3. Java bytecode verification
4. Revision
5. PhD topics
6. Final exam

Example 1: Test.java

```
class Test {
   public static void main(String argv[]) { }
}
```

Example 1: Java Class File (Output of od -An -tx1 Test.class)

```
ca fe ba be 00 03 00 2d 00 0f 07 00 0b 07 00 0d
0a 00 02 00 04 0c 00 07 00 05 01 00 03 28 29 56
01 00 16 28 5b 4c 6a 61 76 61 2f 6c 61 6e 67 2f
53 74 72 69 6e 67 3b 29 56 01 00 06 3c 69 6e 69
74 3e 01 00 04 43 6f 64 65 01 00 0f 4c 69 6e 65
4e 75 6d 62 65 72 54 61 62 6c 65 01 00 0a 53 6f
75 72 63 65 46 69 6c 65 01 00 04 54 65 73 74 01
00 09 54 65 73 74 2e 6a 61 76 61 01 00 10 6a 61
76 61 2f 6c 61 6e 67 2f 4f 62 6a 65 63 74 01 00
04 6d 61 69 6e 00 20 00 01 00 02 00 00 00 00 00
02 00 00 00 07 00 05 00 01 00 08 00 00 00 1d 00
01 00 01 00 00 00 05 2a b7 00 03 b1 00 00 00 01
00 09 00 00 00 06 00 01 00 00 00 01 00 09 00 0e
00 06 00 01 00 08 00 00 00 19 00 00 00 01 00 00
00 01 b1 00 00 00 01 00 09 00 00 00 06 00 01 00
00 00 03 00 01 00 0a 00 00 00 02 00 0c
```

- The classfile format defined in the Sun's JVM spec book
- BCEL's listclass useful for understanding the constant pool
- Generated using Java 45.3

Example 1: Test.class Annotated

```
ca fe ba be     magic

00 03  minor version

00 2d  major version

00 0f  constant pool count
-------------------- constant pool table -------------------------
07 00 0b
07 00 0d
0a 00 02 00 04
0c 00 07 00 05
01 00 03 28 29 56
01 00 16 28 5b 4c 6a 61 76 61 2f 6c 61 6e 67 2f 53 74 72 69 6e 67 3b 29 56
01 00 06 3c 69 6e 69 74 3e
01 00 04 43 6f 64 65
01 00 0f 4c 69 6e 65 4e 75 6d 62 65 72 54 61 62 6c 65
01 00 0a 53 6f 75 72 63 65 46 69 6c 65
01 00 04 54 65 73 74
01 00 09 54 65 73 74 2e 6a 61 76 61
01 00 10 6a 61 76 61 2f 6c 61 6e 67 2f 4f 62 6a 65 63 74
01 00 04 6d 61 69 6e
-------------------- end of constant pool table -------------------------

00 20  access_flags

00 01  this_class

00 02  super_class

00 00  interfaces_count
```

```
00 00  fields_count

00 02  methods_count

------------------ <init> --------------------------------------
00 00  access_flags
00 07  name_index
00 05  desc_index
00 01  attr_count
00 08  attr_name_index
00 00 00 1d  attr_length
00 01  max_stack
00 01  max_locals
00 00 00 05  code_length
2a b7 00 03 b1  code
00 00  exception_table_length
00 01 attr_count
00 09  LineNumberTable
00 00 00 06  attr_length
00 01    line_number_table_length
00 00  start_pc
00 01  line_number
------------------ end of <init> --------------------------------


----------------------------- main  -----------------------------
00 09  access_flags
00 0e name_index
00 06  desc_index
00 01  attr_count
00 08  attr_name_index
00 00 00 19  attr_length
00 00  max_stack
00 01  max_locals
00 00 00 01  code_length
```

```
b1  code
00 00  exception_table_length
00 01  attr_count
00 09  attr_name_index
00 00 00 06  attr_length
00 01  line_number_table_length
00 00  start_pc
00 03  line_number
------------------------------ end of main  ------------------------
00 01  attr_count
00 0a  attr_name_index
00 00 00 02  attr_length
00 0c sourcefile_index
```

Example 1: Test.class Annotated (java listclass -code -constants Test.class, where listclass is from BCEL)

```
class Test extends java.lang.Object
filename Test.class
compiled from Test.java
compiler version 45.3
access flags 32
constant pool 15 entries
ACC_SUPER flag true

Attribute(s):
SourceFile(Test.java)

2 methods:
void <init>()
public static void main(String[])

1)CONSTANT_Class[7](name_index = 11)
2)CONSTANT_Class[7](name_index = 13)
3)CONSTANT_Methodref[10](class_index = 2, name_and_type_index = 4)
4)CONSTANT_NameAndType[12](name_index = 7, signature_index = 5)
5)CONSTANT_utf8[1]("()V")
6)CONSTANT_Utf8[1]("([Ljava/lang/String;)V")
7)CONSTANT_Utf8[1]("<init>")
8)CONSTANT_Utf8[1]("Code")
9)CONSTANT_Utf8[1]("LineNumberTable")
10)CONSTANT_Utf8[1]("SourceFile")
```

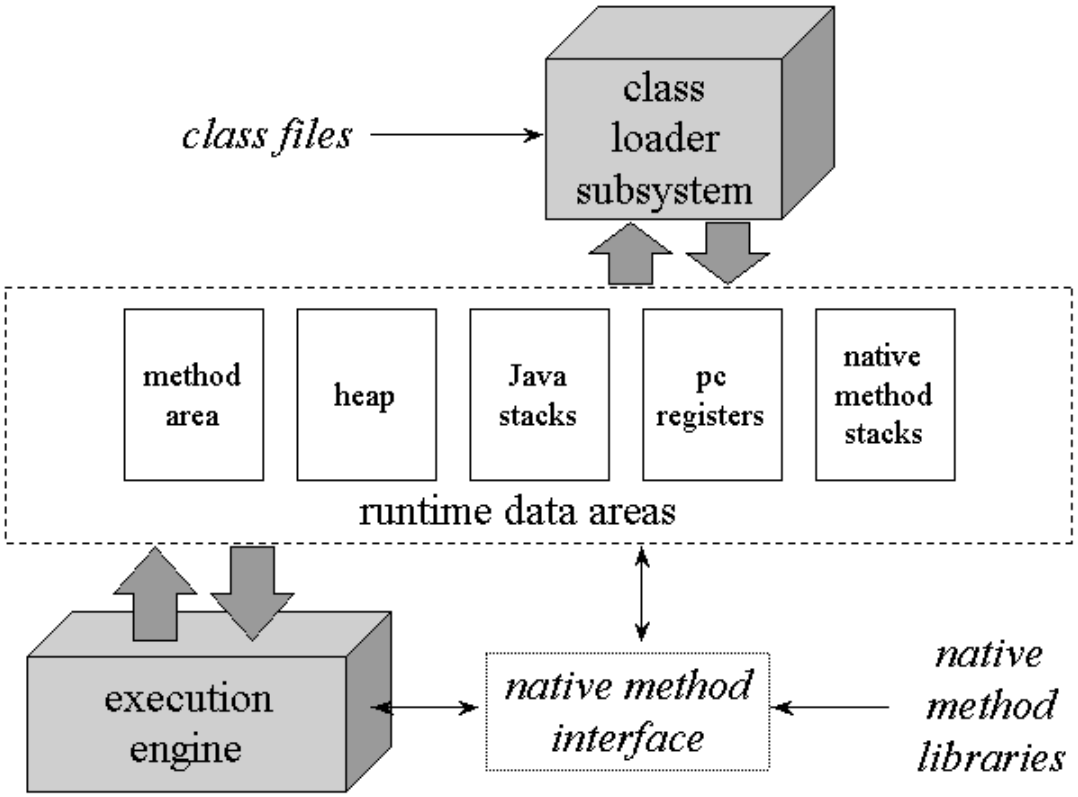
```
11)CONSTANT_Utf8[1]("Test")
12)CONSTANT_Utf8[1]("Test.java")
13)CONSTANT_Utf8[1]("java/lang/Object")
14)CONSTANT_Utf8[1]("main")

void <init>()
Code(max_stack = 1, max_locals = 1, code_length = 5)
0:    aload_0
1:    invokespecial java.lang.Object.<init> ()V (3)
4:    return

Attribute(s) =
LineNumber(0, 1)

public static void main(String[])
Code(max_stack = 0, max_locals = 1, code_length = 1)
0:    return

Attribute(s) =
LineNumber(0, 3)
```

Java Runtime Areas

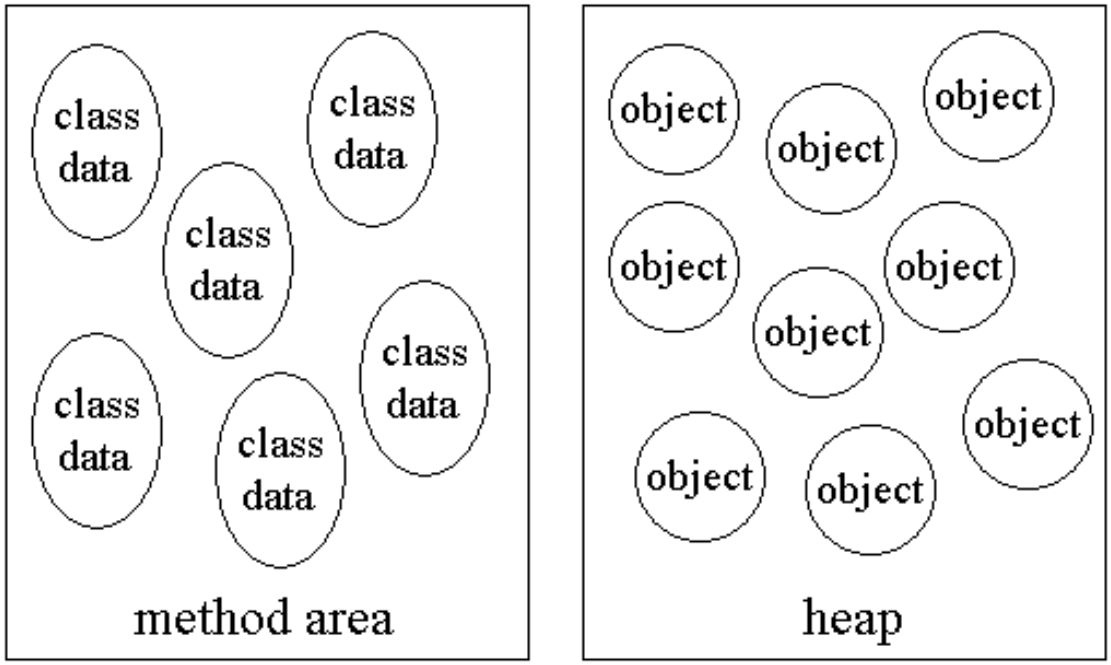
1. pc registers
2. Java stacks (due to multi-threading)
   - Local variables
   - Operand stack
   - Frame data
     - pointer to constant pool
     - information about method return
     - ...
3. Method Area
4. Heap

All JVM diagrams are from Bill Venner's Inside the JVM book

Internal Architecture of JVM

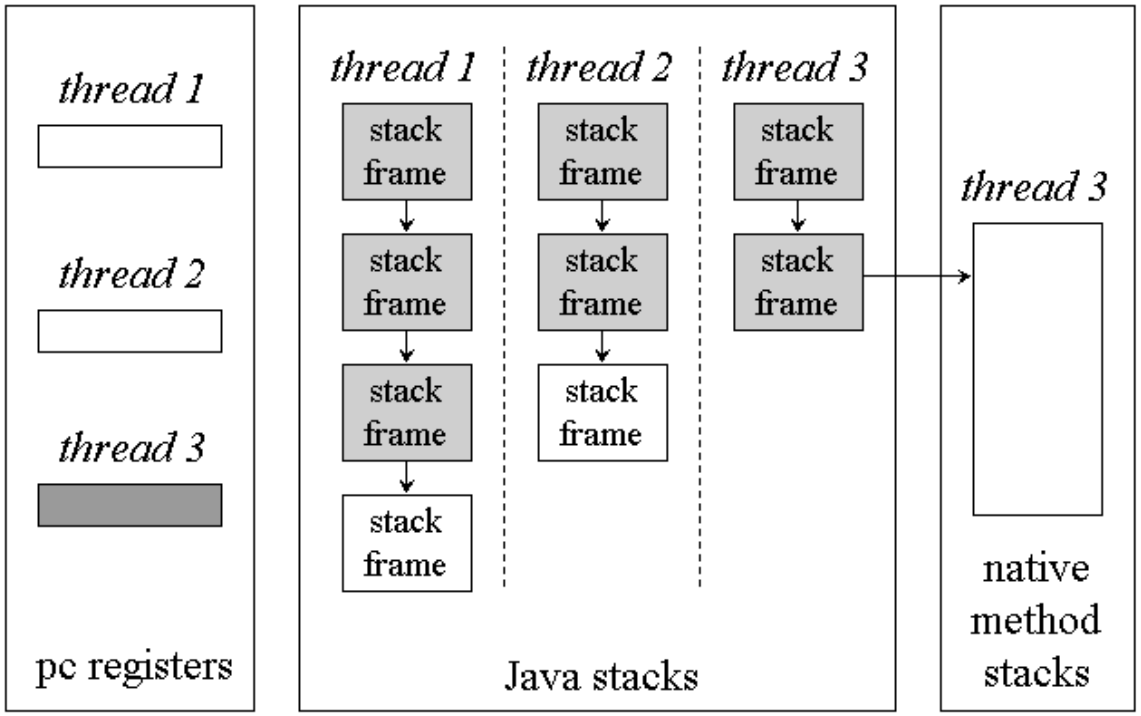


Method Area and Heap



- Arrays are objects

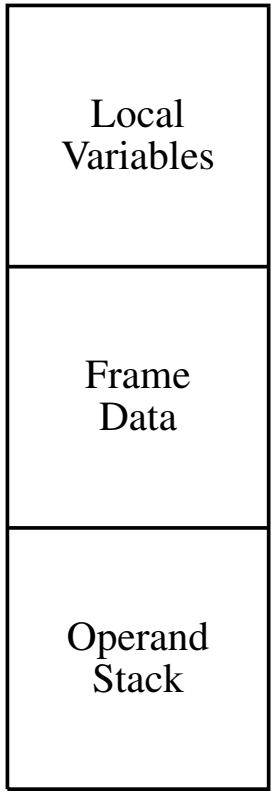
Java Stacks – One Per Thread



Stack Frame

- A new stack frame is created for every method invocation
- Stack frame consists of:
  - Local variables
  - Frame data
  - Operand stack

One Possible Implementation of a Java Frame



Example 2: gcd.vc

```
int i = 5;
int j = 2;
// find the greatest common divisor of two integers
int gcd(int a, int b) {
  if (b == 0)
    return a;
  else
  return gcd(b, a - (a/b) *b);
}
int main() {
  putIntLn(gcd(i, j));
}
```

Example 2: gcd.java

```
import VC.lang.System;

public class gcd {
  static int i = 5;
  static int j = 2;

  public gcd() { }

  int gcd(int a, int b) {
    if (b == 0)
      return a;
    else
    return gcd(b, a - (a/b) *b);
  }

  public static void main(String argv[]) {
    gcd vc$;
    vc$  = new gcd();
    System.putIntLn(vc$.gcd(i, j));
  }
}
```

Example 2: gcd.j

```
.class public gcd
.super java/lang/Object

.field static i I
.field static j I

        ; standard class static initializer
.method static <clinit>()V

        iconst_5
        putstatic gcd/i I
        iconst_2
        putstatic gcd/j I

        ; set limits used by this method
.limit locals 0
.limit stack 1
        return
.end method

        ; standard constructor initializer
.method public <init>()V
.limit stack 1
.limit locals 1
        aload_0
        invokespecial java/lang/Object/<init>()V
        return
.end method

.method  gcd(II)I
.limit stack 5
.limit locals 3
.var 0 is this Lgcd; from Label1 to Label2
.var 1 is a I from Label1 to Label2
.var 2 is b I from Label1 to Label2
```

```
Label1:
.line 10
        iload_2
        ifne Label0
.line 11
        iload_1
        ireturn
Label0:
.line 13
        aload_0
        iload_2
        iload_1
        iload_1
        iload_2
        idiv
        iload_2
        imul
        isub
        invokevirtual gcd/gcd(II)I
Label2:
        ireturn

.end method


.method public static main([Ljava/lang/String;)V
L0:
.var 0 is argv [Ljava/lang/String; from L0 to L1
.var 1 is vc$ Lgcd; from L0 to L1
        new gcd
        dup
        invokespecial gcd/<init>()V
        astore_1
        aload_1
        getstatic gcd/i I
        getstatic gcd/j I
        invokevirtual gcd/gcd(II)I
        invokestatic VC/lang/System/putIntLn(I)V
L1:
```

```
        ; The following return inserted by the VC compiler
        return

        ; set limits used by this method
.limit locals 2
.limit stack 3
.end method
```

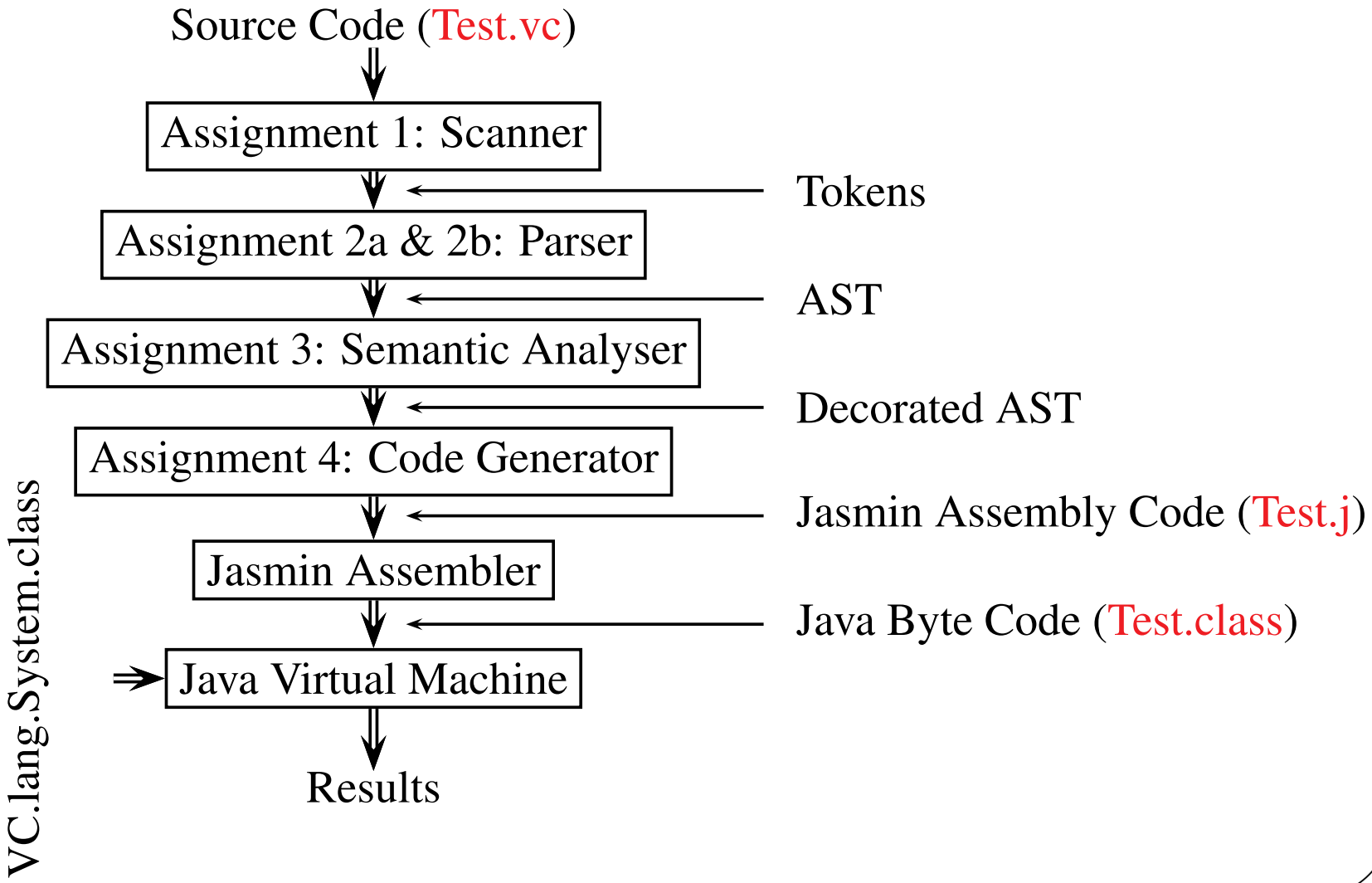
Reading

- Class file format (Sun's JVM Spec Book)
- Runtime areas (JVM Spec & Bill Venners' Inside the JVM book)

Lecture 11: Java Class File Format & Runtime Areas

1. Timetable √
2. Class file format (Sun's JVM Spec Book) √
3. Runtime areas (JVM Spec & Bill Venners' Inside the JVM book) √
4. Java bytecode verification
5. Revision
6. PhD topics
7. Final exam

The VC Compiler

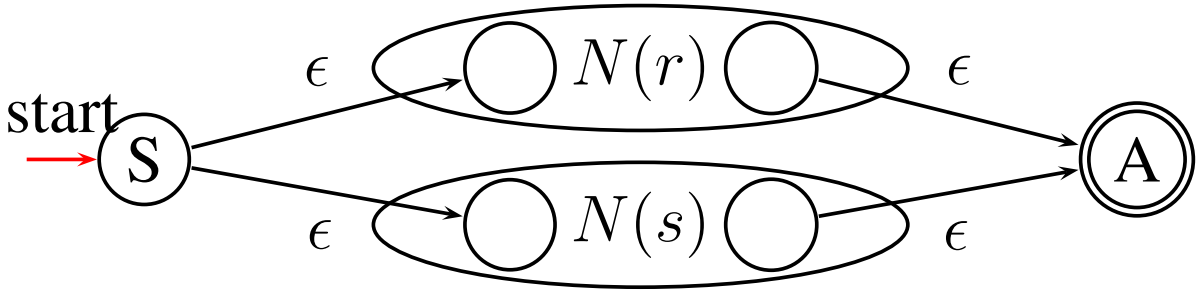


Scanning

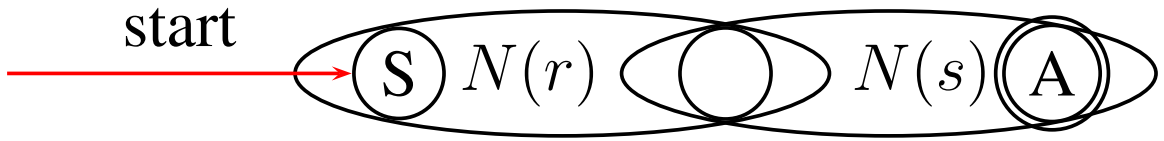
- Theory:
  - Regular expressions (i.e., regular grammars)
  - Finite automata (DFAs and NFAs)
- Writing a scanner
  - Hand coding
  - Scanner generator (JLex and JavaCC)

Thompson's Construction (Cont'd): Slide 79

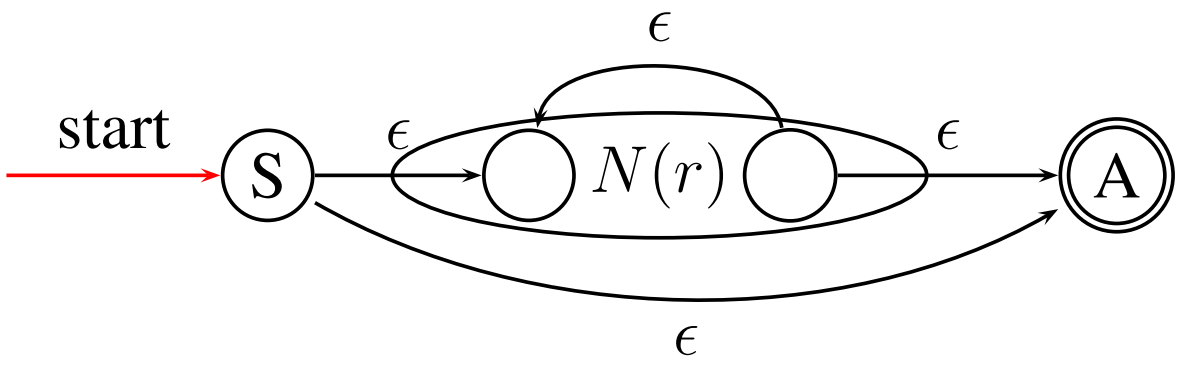
RE $r|s$:



RE rs :



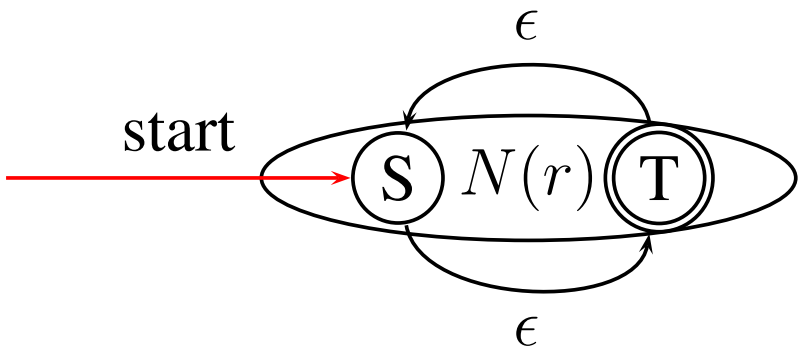
RE r^* :



RE (r) : $N((r))$ is the same as $N(r)$

False RE r^* Rule

RE r^* :



- The FA constructed using this rule may not be correct.
- Try $(a^*b)^*$!
- The ϵ edges in Thompson's construction ensure that the construction is correct, and are removed when the NFA is converted to a DFA

Parsing

- Theory:
  - Context-free grammars
  - First, Follow and Select sets
- Writing a parser
  - Recursive-descent
  - Table-driven LL(1)
  - Parser generators (JavaCUP and JavaCC)

LR(k) not covered this year

Context-Sensitive Analysis

- Theory:
  - Attribute grammars
  - Evaluation of attributes
  - Scoping
  - Type rules
- Writing a context-sensitive analyser
  - Identification: Relate the use of an identifier to its declaration
  - Type Checking: Enforce the language's type rules

Code Generation

- Theory:
  - Syntax-directed translation
  - Code templates
- Writing a code generator
  - JVM
  - Jasmin assembly language

PhD Projects

- Some current PhD projects:
  - Compiler techniques for heterogeneous architectures (CPUs + TPUs/GPUs/FPGAs)
  - Concurrent programming (e.g., programming model and OO)
  - Program analysis for safety and security
- PhD topics
  - Static and dynamic analysis for bug detection
  - Security analysis of Android apps
  - Language-based safety and security
  - Languages and compilers for multicore computing (e.g., GPUs)
  - Machine learning guided security analysis

Final Exam

- Online
- 3 hours
- The answers must be submitted using give.
- The 2021 exam paper and solution:

```
1. Login to a CSE computer
2. Type:
cp ~cs3131/Exams/2021paper.zip your-directory
```

Remember to Complete myExperience Survey for COMP3131/9102

Good luck with your studies!