Automata and Formal Languages (9)

Email:	christian.urban at kcl.ac.uk
Office:	S1.27 (1st floor Strand Building)
Slides:	KEATS (also home work is there)

Imagine the following situation: You talk to somebody and you find out that she/he has implemented a compiler.

What is your reaction? Check all that apply.

Imagine the following situation: You talk to somebody and you find out that she/he has implemented a compiler.

What is your reaction? Check all that apply.

- □ You think she/he is God
- □ Überhacker
- □ superhuman
- □ wizard
- □ supremo

While-Language

$$
\begin{array}{lcl}
Stmt & \rightarrow & \text{skip} \\
 & | & Id := AExp \\
 & | & \text{if } BExp \text{ then } Block \text{ else } Block \\
 & | & \text{while } BExp \text{ do } Block \\
 & | & \text{write } Id \\
Stmts & \rightarrow & Stmt \;;\; Stmts \\
 & | & Stmt \\
Block & \rightarrow & \{Stmts\} \\
 & | & Stmt \\
AExp & \rightarrow & \ldots \\
BExp & \rightarrow & \ldots
\end{array}
$$

Fibonacci Numbers

```
/* Fibonacci Program
   input: n
   output: fib_res */

n := 90;
minus1 := 0;
minus2 := 1;
temp := 0;
while n > 0 do {
       temp := minus2;
       minus2 := minus1 + minus2;
       minus1 := temp;
       n := n - 1
};
fib_res := minus2;
write fib_res
```

Interpreter

$$
\begin{aligned}
\text{eval}(n, E) &\overset{\text{def}}{=} n \\
\text{eval}(x, E) &\overset{\text{def}}{=} E(x) \quad \text{lookup } x \text{ in } E \\
\text{eval}(a_1 + a_2, E) &\overset{\text{def}}{=} \text{eval}(a_1, E) + \text{eval}(a_2, E) \\
\text{eval}(a_1 - a_2, E) &\overset{\text{def}}{=} \text{eval}(a_1, E) - \text{eval}(a_2, E) \\
\text{eval}(a_1 * a_2, E) &\overset{\text{def}}{=} \text{eval}(a_1, E) * \text{eval}(a_2, E)
\end{aligned}
$$

$$
\begin{aligned}
\text{eval}(a_1 = a_2, E) &\overset{\text{def}}{=} \text{eval}(a_1, E) = \text{eval}(a_2, E) \\
\text{eval}(a_1 \mathrel{!=} a_2, E) &\overset{\text{def}}{=} \neg(\text{eval}(a_1, E) = \text{eval}(a_2, E)) \\
\text{eval}(a_1 < a_2, E) &\overset{\text{def}}{=} \text{eval}(a_1, E) < \text{eval}(a_2, E)
\end{aligned}
$$

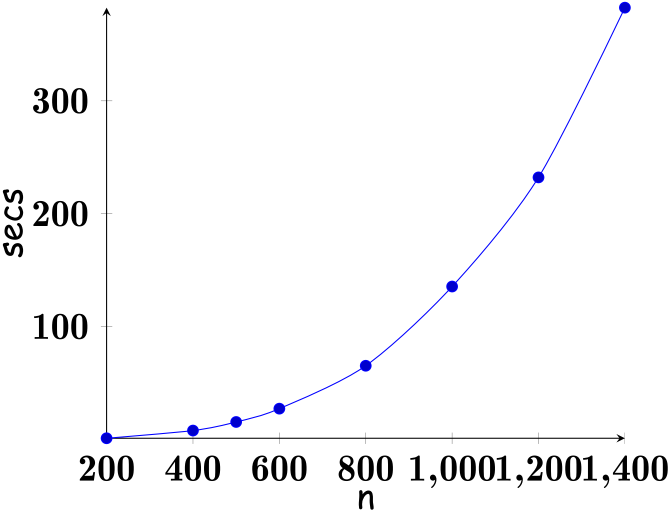
Interpreter (2)

$$\begin{aligned}
\mathsf{eval}(\mathsf{skip}, E) &\overset{\mathsf{def}}{=} E\\
\mathsf{eval}(x := a, E) &\overset{\mathsf{def}}{=} E(x \mapsto \mathsf{eval}(a, E))\\
\mathsf{eval}(\mathsf{if}\ b\ \mathsf{then}\ cs_1\ \mathsf{else}\ cs_2, E) &\overset{\mathsf{def}}{=}\\
&\quad \mathsf{if}\ \mathsf{eval}(b, E)\ \mathsf{then}\ \mathsf{eval}(cs_1, E)\\
&\quad \phantom{\mathsf{if}\ \mathsf{eval}(b, E)}\ \mathsf{else}\ \mathsf{eval}(cs_2, E)\\
\mathsf{eval}(\mathsf{while}\ b\ \mathsf{do}\ cs, E) &\overset{\mathsf{def}}{=}\\
&\quad \mathsf{if}\ \mathsf{eval}(b, E)\\
&\quad \mathsf{then}\ \mathsf{eval}(\mathsf{while}\ b\ \mathsf{do}\ cs, \mathsf{eval}(cs, E))\\
&\quad \mathsf{else}\ E\\
\mathsf{eval}(\mathsf{print}\ x, E) &\overset{\mathsf{def}}{=} \{\ \mathsf{println}(E(x))\ ;\ E\ \}
\end{aligned}$$

Test Program

```
start := 1;
x := start;
y := start;
z := start;
while 0 < x do {
 while 0 < y do {
  while 0 < z do {
    z := z - 1
  };
  z := start;
  y := y - 1
 };
 y := start;
 x := x - 1
};
write x;
write y;
write z
```

Interpreted Code



Java Virtual Machine

- introduced in 1995
- is a stack-based VM (like Postscript, CLR of .Net)
- contains a JIT compiler
- many languages take advantage of the infrastructure (JRE)
- languages compiled to the JVM: Scala, Clojure...
- garbage collected

Compiling AExps

1 + 2

```
ldc 1
ldc 2
iadd
```

Compiling AExps

1 + 2 + 3

```
ldc 1
ldc 2
iadd
ldc 3
iadd
```

Compiling AExps

$1 + (2 + 3)$

```
ldc 1
ldc 2
ldc 3
iadd
iadd
```

Compiling AExps

1 + (2 + 3)

```
ldc 1
ldc 2
ldc 3
iadd
iadd
```

dadd, fadd, ladd, . . .

Compiling AExps

$$\begin{aligned}
\text{compile}(n) &\overset{\text{def}}{=} \text{ldc } n\\
\text{compile}(a_1 + a_2) &\overset{\text{def}}{=}\\
&\quad \text{compile}(a_1)\ @\ \text{compile}(a_2)\ @\ \text{iadd}\\
\text{compile}(a_1 - a_2) &\overset{\text{def}}{=}\\
&\quad \text{compile}(a_1)\ @\ \text{compile}(a_2)\ @\ \text{isub}\\
\text{compile}(a_1 * a_2) &\overset{\text{def}}{=}\\
&\quad \text{compile}(a_1)\ @\ \text{compile}(a_2)\ @\ \text{imul}
\end{aligned}$$

Compiling AExps

$$\begin{aligned}
\text{compile}(n) &\overset{\text{def}}{=} \text{ldc } n \\
\text{compile}(a_1 + a_2) &\overset{\text{def}}{=} \\
&\quad \text{compile}(a_1) \mathbin{@} \text{compile}(a_2) \mathbin{@} \text{iadd} \\
\text{compile}(a_1 - a_2) &\overset{\text{def}}{=} \\
&\quad \text{compile}(a_1) \mathbin{@} \text{compile}(a_2) \mathbin{@} \text{isub} \\
\text{compile}(a_1 * a_2) &\overset{\text{def}}{=} \\
&\quad \text{compile}(a_1) \mathbin{@} \text{compile}(a_2) \mathbin{@} \text{imul}
\end{aligned}$$

Compiling AExps

1 + 2 * 3 + (4 - 3)

```
ldc 1
ldc 2
ldc 3
imul
ldc 4
ldc 3
isub
iadd
iadd
```

Variables

$$x := 5 + y * 2$$

Variables

$$x := 5 + y * 2$$

- lookup: iload $number$
- store: istore $number$

Variables

$$x := 5 + y * 2$$

- lookup: iload $number$
- store: istore $number$

during compilation we have to maintain a map between our identifiers and the java bytecode numbers

compile(a, E)

Compiling AExps

$$\textsf{compile}(n, E) \overset{\text{def}}{=} \textsf{ldc}\ n$$

$$\textsf{compile}(a_1 + a_2, E) \overset{\text{def}}{=}$$
$$\quad \textsf{compile}(a_1, E)\ @\ \textsf{compile}(a_2.E)\ @\ \textsf{iadd}$$

$$\textsf{compile}(a_1 - a_2, E) \overset{\text{def}}{=}$$
$$\quad \textsf{compile}(a_1, E)\ @\ \textsf{compile}(a_2, E)\ @\ \textsf{isub}$$

$$\textsf{compile}(a_1 * a_2, E) \overset{\text{def}}{=}$$
$$\quad \textsf{compile}(a_1, E)\ @\ \textsf{compile}(a_2, E)\ @\ \textsf{imul}$$

$$\textsf{compile}(x, E) \overset{\text{def}}{=} \textsf{iload}\ E(x)$$

Compiling AExps

$$\text{compile}(n, E) \overset{\text{def}}{=} \text{ldc}\ n$$

$$\text{compile}(a_1 + a_2, E) \overset{\text{def}}{=}$$
$$\text{compile}(a_1, E)\ @\ \text{compile}(a_2.E)\ @\ \text{iadd}$$

$$\text{compile}(a_1 - a_2, E) \overset{\text{def}}{=}$$
$$\text{compile}(a_1, E)\ @\ \text{compile}(a_2, E)\ @\ \text{isub}$$

$$\text{compile}(a_1 * a_2, E) \overset{\text{def}}{=}$$
$$\text{compile}(a_1, E)\ @\ \text{compile}(a_2, E)\ @\ \text{imul}$$

$$\text{compile}(x, E) \overset{\text{def}}{=} \text{iload}\ E(x)$$

Compiling Statements

We return a list of instructions and an environment for the variables

$$\text{compile}(\text{skip}, E) \quad \overset{\text{def}}{=} (Nil, E)$$

$$\text{compile}(x := a, E) \overset{\text{def}}{=}$$
$$(\text{compile}(a, E) \ @ \ \text{istore}\ index, E(x \mapsto index))$$

where $index$ is $E(x)$ if it is already defined, or if it is not then the largest index not yet seen

Compiling AExps

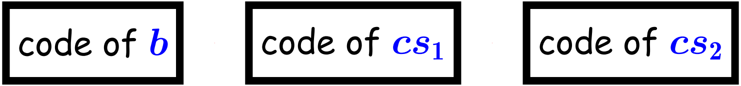
$x := x + 1$

```
iload $n_x$
ldc 1
iadd
istore $n_x$
```

where n_x is the number corresponding to the variable x

Compiling Ifs

if b else cs_1 then cs_2



code of b

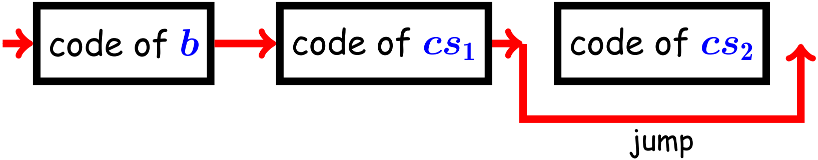
code of cs_1

code of cs_2

Compiling Ifs

if b else cs_1 then cs_2

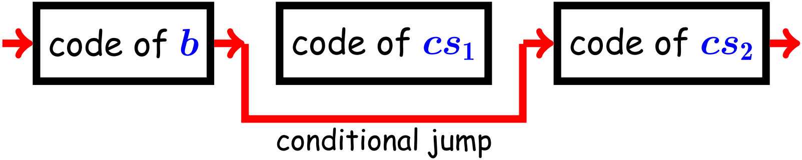
Case True:



Compiling Ifs

if b else cs_1 then cs_2

Case False:



Conditional Jumps

- if_icmpeq $label$ if two ints are equal, then jump
- if_icmpne $label$ if two ints aren't equal, then jump
- if_icmpge $label$ if one int is greater or equal then another, then jump

  ...

Conditional Jumps

- if_icmpeq *label* if two ints are equal, then jump
- if_icmpne *label* if two ints aren't equal, then jump
- if_icmpge *label* if one int is greater or equal then another, then jump

...

```
L_1:
   if_icmpeq L_2
   iload 1
   ldc 1
   iadd
   if_icmpeq L_1
L_1:
```

Compiling BExps

$a_1 = a_2$

```
iload n_x
ldc 1
iadd
istore n_x
```

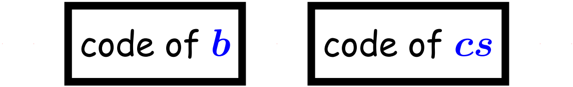
Compiling Ifs

if b then cs_1 else cs_2

iload n_x
ldc 1
iadd
istore n_x

Compiling Whiles

while b do cs



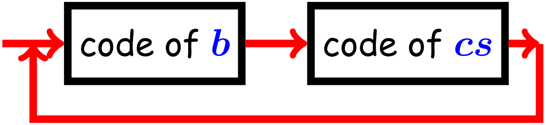
code of b

code of cs

Compiling Whiles

while b do cs

Case True:



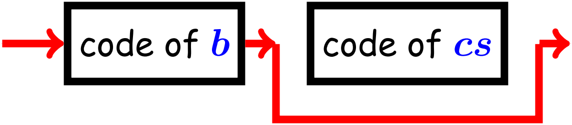
code of b

code of cs

Compiling Whiles

while b do cs

Case False:



code of b

code of cs

Compiling Whiles

while b do cs

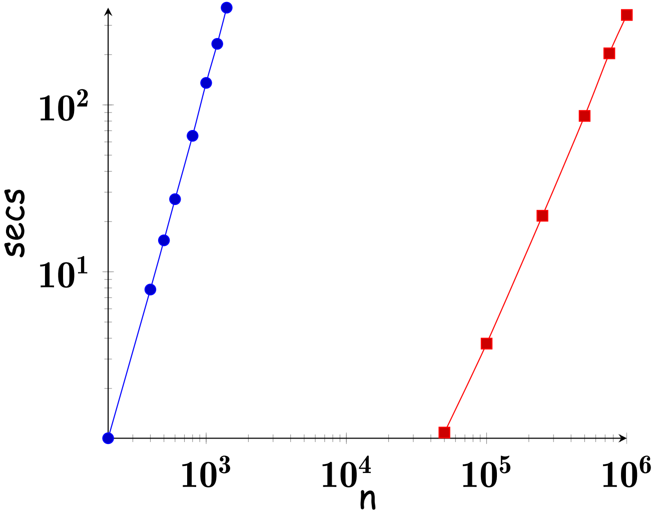
iload n_x
ldc 1
iadd
istore n_x

Compiling Writes

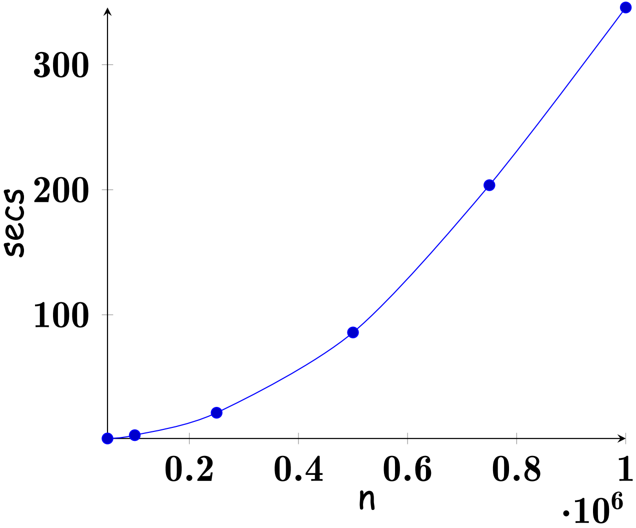
write x

```
iload $n_x$
ldc 1
iadd
istore $n_x$
```

Compiled vs. Interpreted Code



Compiled Code


secs
n
$\cdot 10^6$