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)

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

```
1  /* Fibonacci Program
2     input: n
3     output: fib_res */
4
5  n := 90;
6  minus1 := 0;
7  minus2 := 1;
8  temp := 0;
9  while n > 0 do  {
10         temp := minus2;
11         minus2 := minus1 + minus2;
12         minus1 := temp;
13         n := n - 1
14 };
15 fib_res := minus2;
16 write fib_res
```

Interpreter

Interpreting a List of Tokens

The lexer cannot prevent errors like

$$ \ldots <p> \ldots \ldots </p>$$

or

$$ \ldots <b>$$

Parser Combinators

Parser combinators:

$$\underbrace{\text{list of tokens}}_{\text{input}} \Rightarrow \underbrace{\text{set of (parsed input, unparsed input)}}_{\text{output}}$$

- sequencing
- alternative
- semantic action

Alternative parser (code $p \mid\mid q$)

- apply p and also q; then combine the outputs

$$p(\text{input}) \cup q(\text{input})$$

Sequence parser (code $p \sim q$)

- apply first p producing a set of pairs
- then apply q to the unparsed parts
- then combine the results:
  ((output$_1$, output$_2$), unparsed part)

$$\{((o_1, o_2), u_2) \mid (o_1, u_1) \in p(\text{input}) \wedge (o_2, u_2) \in q(u_1)\}$$

Function parser (code $p \Longrightarrow f$)

- apply p producing a set of pairs
- then apply the function f to each first component

$$\{(f(o_1), u_1) \mid (o_1, u_1) \in p(\text{input})\}$$

Function parser (code $p \Longrightarrow f$)

- apply p producing a set of pairs
- then apply the function f to each first component

$$\{(f(o_1), u_1) \mid (o_1, u_1) \in p(\text{input})\}$$

f is the semantic action ("what to do with the parsed input")

Token parser:

- if the input is

$$tok_1 :: tok_2 :: \ldots :: tok_n$$

then return

$$\{(tok_1,\ tok_2 :: \ldots :: tok_n)\}$$

or

$$\{\}$$

if tok_1 is not the right token we are looking for

Number-Token parser:

- if the input is

$$num\_tok(42) :: tok_2 :: \ldots :: tok_n$$

then return

$$\{(42,\ tok_2 :: \ldots :: tok_n)\}$$

or

$$\{\}$$

if tok_1 is not the right token we are looking for

Number-Token parser:

- if the input is

$$num\_tok(42) :: tok_2 :: \ldots :: tok_n$$

then return

$$\{(42,\ tok_2 :: \ldots :: tok_n)\}$$

or

$$\{\}$$

if tok_1 is not the right token we are looking for

list of tokens $\Rightarrow$ set of (int, list of tokens)

- if the input is

$$num\_tok(42) :: num\_tok(3) :: tok_3 :: \ldots :: tok_n$$

and the parser is

$$ntp \sim ntp$$

the successful output will be

$$\{((42, 3),\ tok_2 :: \ldots :: tok_n)\}$$

- if the input is

$$num\_tok(42) :: num\_tok(3) :: tok_3 :: \ldots :: tok_n$$

and the parser is

$$ntp \sim ntp$$

the successful output will be

$$\{((42, 3),\ tok_2 :: \ldots :: tok_n)\}$$

Now we can form

$$(ntp \sim ntp) \Longrightarrow f$$

where f is the semantic action ("what to do with the pair")

Semantic Actions

Addition

$$T \sim + \sim E \Longrightarrow \underbrace{f((x, y), z) \Rightarrow x + z}_{\text{semantic action}}$$

Semantic Actions

Addition

$$T \sim + \sim E \Longrightarrow \underbrace{f((x, y), z) \Rightarrow x + z}_{\text{semantic action}}$$

Multiplication

$$F \sim * \sim T \Longrightarrow f((x, y), z) \Rightarrow x * z$$

Semantic Actions

Addition

$$T \sim + \sim E \Longrightarrow \underbrace{f((x, y), z) \Rightarrow x + z}_{\text{semantic action}}$$

Multiplication

$$F \sim * \sim T \Longrightarrow f((x, y), z) \Rightarrow x * z$$

Parenthesis

$$(\sim E \sim) \Longrightarrow f((x, y), z) \Rightarrow y$$

Types of Parsers

- **Sequencing**: if p returns results of type T, and q results of type S, then $p \sim q$ returns results of type

$$T \times S$$

Types of Parsers

- **Sequencing**: if p returns results of type T, and q results of type S, then $p \sim q$ returns results of type

$$T \times S$$

- **Alternative**: if p returns results of type T then q must also have results of type T, and $p \,||\, q$ returns results of type

$$T$$

Types of Parsers

- **Sequencing**: if p returns results of type T, and q results of type S, then $p \sim q$ returns results of type

$$T \times S$$

- **Alternative**: if p returns results of type T then q must also have results of type T, and $p \,||\, q$ returns results of type

$$T$$

- **Semantic Action**: if p returns results of type T and f is a function from T to S, then $p \Longrightarrow f$ returns results of type

$$S$$

Input Types of Parsers

- input: list of tokens
- output: set of (output_type, list of tokens)

Input Types of Parsers

- input: list of tokens
- output: set of (output_type, list of tokens)

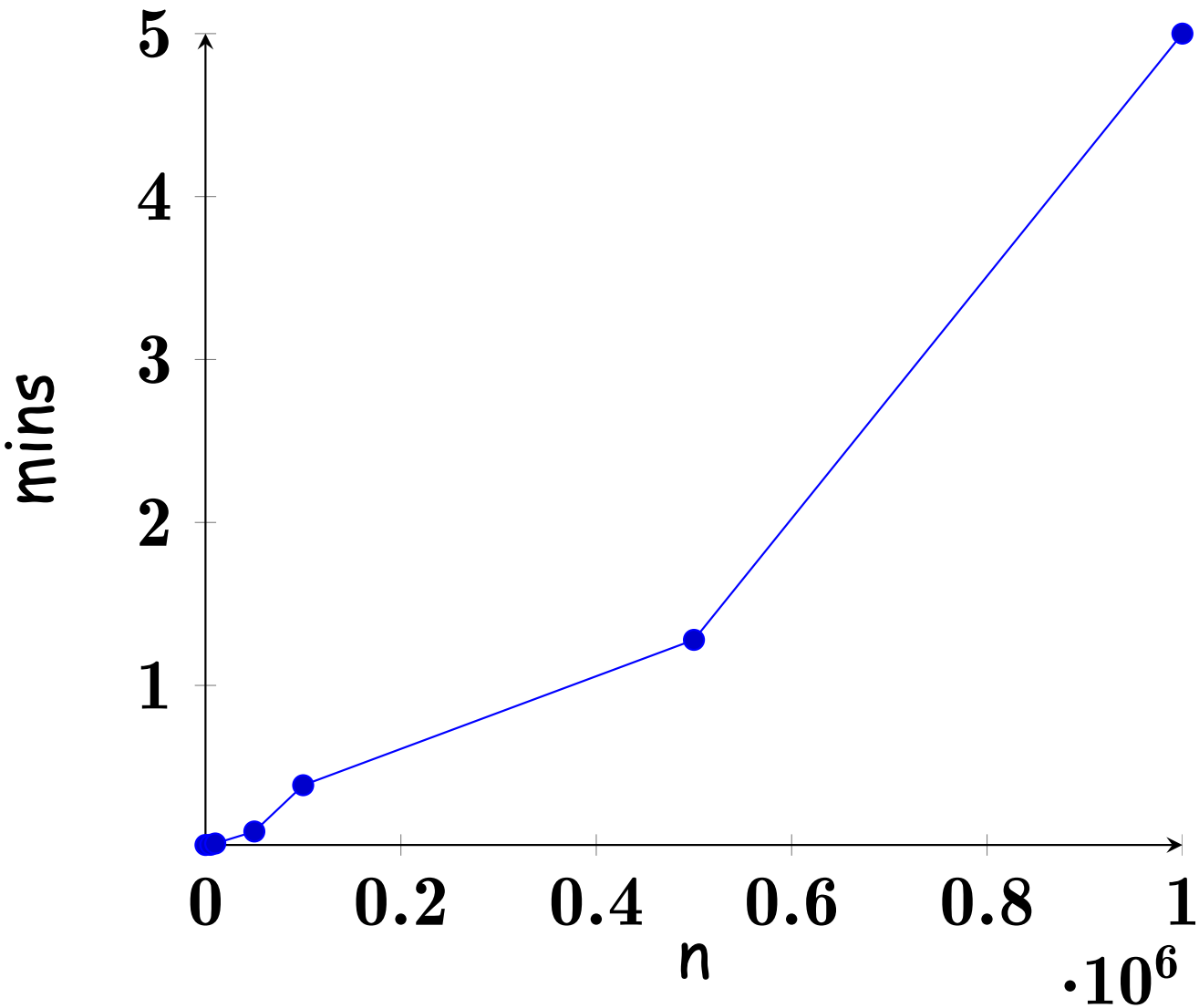
actually it can be any input type as long as it is a kind of sequence (for example a string)

Scannerless Parsers

- input: string
- output: set of (output_type, string)

but lexers are better when whitespaces or comments need to be filtered out

Compiled vs. Interpreted Code



Compiled vs. Interpreted Code