## **Compilers and Formal Languages**

Email: christian.urban at kcl.ac.uk Office: N7.07 (North Wing, Bush House) Slides: KEATS (also home work is there)

# **Compilers & Boeings 777**

First flight in 1994. They want to achieve triple redundancy in hardware faults.

They compile 1 Ada program to

- Intel 80486
- Motorola 68040 (old Macintosh's)
- AMD 29050 (RISC chips used often in laser printers)

using 3 independent compilers.

**Compilers & Boeings 777** 

First flight in 1994. They want to achieve triple redundancy in hardware faults.

They compile 1 Ada program to

- Intel 80486
- Motorola 68040 (old Macintosh's)
- AMD 29050 (RISC chips used often in laser printers)

using 3 independent compilers.

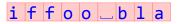
Airbus uses C and static analysers. Recently started using CompCert.



- verified a microkernel operating system (≈8000 lines of C code)
- US DoD has competitions to hack into drones; they found that the isolation guarantees of seL4 hold up
- CompCert and seL4 sell their code

## **POSIX Matchers**

• Longest match rule ("maximal munch rule"): The longest initial substring matched by any regular expression is taken as the next token.



• Rule priority: For a particular longest initial substring, the first regular expression that can match determines the token.

i f \_ b l a

## **POSIX Matchers**

• Longest match rule ("maximal munch rule"): The longest initial substring matched by any regular expression is taken as the next token.



• Rule priority: For a particular longest initial substring, the first regular expression that can match determines the token.

if\_bla

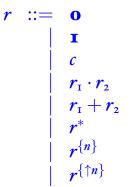
Kuklewicz: most POSIX matchers are buggy http://www.haskell.org/haskellwiki/Regex\_Posix

def der  $c(\mathbf{0})$ 0 def der  $c(\mathbf{I})$ 0  $\stackrel{\text{def}}{=}$ der c(d) $\stackrel{\text{def}}{=}$ der c  $(r_1 + r_2)$ def der c  $(r_1 \cdot r_2)$  $\stackrel{\text{def}}{=}$ der c  $(r^*)$ *der*  $c(r^{\{n\}})$ der c  $(r^{\uparrow n})$ 

if c = d then **I** else **0**  $(der \ c \ r_1) + (der \ c \ r_2)$ if nullable( $r_{I}$ ) then  $((der \ c \ r_1) \cdot r_2) + (der \ c \ r_2)$ else (der c  $r_1$ ) ·  $r_2$  $(der \ c \ r) \cdot (r^*)$  $\stackrel{\text{def}}{=}$  if n = 0 then **0** else if nullable(r) then (der c r)  $\cdot$  ( $r^{\{\uparrow n-1\}}$ ) else (der c r)  $\cdot$  ( $r^{\{n-1\}}$ )  $\stackrel{\text{def}}{=}$  if n = 0 then **0** else (der c r)  $\cdot$  ( $r^{\{\uparrow n-1\}}$ )

### **Proofs about Rexps**

Remember their inductive definition:



If we want to prove something, say a property P(r), for all regular expressions r then ...

## **Proofs about Rexp (2)**

- *P* holds for **0**, **1** and **c**
- *P* holds for  $r_1 + r_2$  under the assumption that *P* already holds for  $r_1$  and  $r_2$ .
- *P* holds for  $r_1 \cdot r_2$  under the assumption that *P* already holds for  $r_1$  and  $r_2$ .
- *P* holds for *r*<sup>\*</sup> under the assumption that *P* already holds for *r*.

...

## **Proofs about Strings**

If we want to prove something, say a property P(s), for all strings *s* then ...

- *P* holds for the empty string, and
- *P* holds for the string *c*::*s* under the assumption that *P* already holds for *s*

## **Correctness of the Matcher**

• We want to prove

#### *matches* r s if and only if $s \in L(r)$

where *matches*  $r s \stackrel{\text{def}}{=} nullable(ders s r)$ 

## Correctness of the Matcher

• We want to prove

*matches* r s if and only if  $s \in L(r)$ 

where *matches*  $r s \stackrel{\text{def}}{=} nullable(ders s r)$ 

• We can do this, if we know

 $L(der \ c \ r) = Der \ c \ (L(r))$ 

### **Some Lemmas**

- Der c  $(A \cup B) = (Der c A) \cup (Der c B)$
- If [] ∈ A then
  Der c (A @ B) = (Der c A) @ B ∪ (Der c B)
- If []  $\notin A$  then Der c (A @ B) = (Der c A) @ B
- Der  $c(A^*) = (Der c A) @A^*$

(interesting case)

Why?

Why does *Der*  $c(A^*) = (Der c A) @A^*$  hold?

$$Der c (A^*) = Der c (A^* - \{[]\})$$
  
= Der c ((A - {[]})@A^\*)  
= (Der c (A - {[]}))@A^\*  
= (Der c A)@A^\*

using the facts *Der c*  $A = Der c (A - \{[]\})$  and  $(A - \{[]\}) @A^* = A^* - \{[]\}$ 

## **POSIX Spec**

$$[] \in \mathbf{I} \to Empty \qquad c \in c \to Char(c)$$

$$\frac{s \in r_{\text{\tiny I}} \to v}{s \in r_{\text{\tiny I}} + r_2 \to Left(v)}$$

$$\frac{s \in r_2 \to v \quad s \notin L(r_1)}{s \in r_1 + r_2 \to Right(v)}$$

$$\frac{s_{\mathrm{I}} \in r_{\mathrm{I}} \to v_{\mathrm{I}}}{s_{2} \in r_{2} \to v_{2}}$$
$$\neg (\exists s_{3}s_{4} \cdot s_{3} \neq [] \land s_{3} @ s_{4} = s_{2} \land s_{\mathrm{I}} @ s_{3} \in L(r_{\mathrm{I}}) \land s_{4} \in L(r_{2}))$$
$$s_{\mathrm{I}} @ s_{2} \in r_{\mathrm{I}} \cdot r_{2} \to Seq(v_{\mathrm{I}}, v_{2})$$

...

## Sulzmann & Lu Paper

• I have no doubt the algorithm is correct — the problem is I do not believe their proof.

"How could I miss this? Well, I was rather careless when stating this Lemma :) Great example how formal machine checked proofs

(and proof assistants) can help to spot flawed reasoning steps."

## Sulzmann & Lu Paper

• I have no doubt the algorithm is correct — the problem is I do not believe their proof.

"How could I miss this? Well, I was rather careless when stating this Lemma :) Great example how formal machine checked proofs

(and proof assistants) can help to spot flawed reasoning steps."

## Sulzmann & Lu Paper

• I have no doubt the algorithm is correct — the problem is I do not believe their proof.

"How could I miss this? Well I was rather careless

Great example how formal machine checked proofs (and proof assistants) can help to spot flawed reasoning steps."