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.

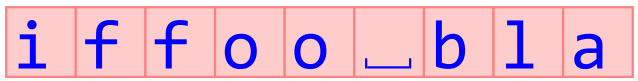
Airbus uses C and static analysers.
Recently started using CompCert.

seL4 / Isabelle

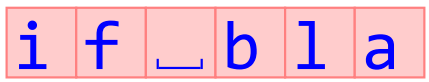
- verified a microkernel operating system ($\approx$8000 lines of C code)
- US DoD has competitions to hack into drones; they found that the isolation guarantees of seL4 hold up
- CompCert and seL4 sell their code

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.



POSIX Matchers

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

i	f	f	o	o	␣	b	l	a

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

i	f	␣	b	l	a

Kuklewicz: most POSIX matchers are buggy

http://www.haskell.org/haskellwiki/Regex_Posix

$$
\begin{array}{lcl}
der\ c\ (\mathbf{0}) & \stackrel{\text{def}}{=} & \mathbf{0} \\
der\ c\ (\mathbf{1}) & \stackrel{\text{def}}{=} & \mathbf{0} \\
der\ c\ (d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \mathbf{1} \text{ else } \mathbf{0} \\
der\ c\ (r_1 + r_2) & \stackrel{\text{def}}{=} & (der\ c\ r_1) + (der\ c\ r_2) \\
der\ c\ (r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } nullable(r_1) \\
 & & \text{then } ((der\ c\ r_1) \cdot r_2) + (der\ c\ r_2) \\
 & & \text{else } (der\ c\ r_1) \cdot r_2 \\
der\ c\ (r^*) & \stackrel{\text{def}}{=} & (der\ c\ r) \cdot (r^*) \\
der\ c\ (r^{\{n\}}) & \stackrel{\text{def}}{=} & \text{if } n = 0 \text{ then } \mathbf{0} \\
 & & \text{else if } nullable(r) \text{ then } (der\ c\ r) \cdot (r^{\{\uparrow n-1\}}) \\
 & & \text{else } (der\ c\ r) \cdot (r^{\{n-1\}}) \\
der\ c\ (r^{\{\uparrow n\}}) & \stackrel{\text{def}}{=} & \text{if } n = 0 \text{ then } \mathbf{0} \\
 & & \text{else } (der\ c\ r) \cdot (r^{\{\uparrow n-1\}})
\end{array}
$$

Proofs about Rexps

Remember their inductive definition:

$$
\begin{array}{rcl}
r & ::= & \mathbf{0} \\
 & | & \mathbf{1} \\
 & | & c \\
 & | & r_1 \cdot r_2 \\
 & | & r_1 + r_2 \\
 & | & r^* \\
 & | & r^{\{n\}} \\
 & | & r^{\{\uparrow n\}}
\end{array}
$$

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

Proofs about Rexp (2)

- P holds for $\mathbf{0}$, $\mathbf{1}$ 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^* 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 $[] \in A$ then

  $$Der\ c\ (A \mathbin{@} B) =$$
  $$(Der\ c\ A) \mathbin{@} B \ \cup\ (Der\ c\ B)$$

- If $[] \notin A$ then

  $$Der\ c\ (A \mathbin{@} B) = (Der\ c\ A) \mathbin{@} B$$

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

  (interesting case)

Why?

Why does

$Der\ c\ (A^*) = (Der\ c\ A) @ A^*$

hold?

$$\begin{array}{lcl} Der\ c\ (A^*) & = & Der\ c\ (A^* - \{[]\}) \\ & = & Der\ c\ ((A - \{[]\}) @ A^*) \\ & = & (Der\ c\ (A - \{[]\})) @ A^* \\ & = & (Der\ c\ A) @ A^* \end{array}$$

using the facts

$Der\ c\ A = Der\ c\ (A - \{[]\})$ and

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

POSIX Spec

$$\frac{}{[] \in \mathbf{1} \rightarrow Empty}$$

$$\frac{}{c \in c \rightarrow Char(c)}$$

$$\frac{s \in r_1 \rightarrow v}{s \in r_1 + r_2 \rightarrow Left(v)}$$

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

$$\frac{\begin{array}{l} s_1 \in r_1 \rightarrow v_1 \\ s_2 \in r_2 \rightarrow v_2 \\ \neg(\exists s_3\, s_4.\ s_3 \neq [] \wedge s_3@s_4 = s_2 \wedge s_1@s_3 \in L(r_1) \wedge s_4 \in L(r_2)) \end{array}}{s_1@s_2 \in r_1 \cdot r_2 \rightarrow Seq(v_1, v_2)}$$

...

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

Lemma 3 (Projection and Injection). *Let r be a regular expression, l a letter and v a parse tree.*

1. *If $\vdash v : r$ and $|v| = lw$ for some word w, then $\vdash proj_{(r,l)}\ v : r\backslash l$.*
2. *If $\vdash v : r\backslash l$ then $(proj_{(r,l)} \circ inj_{r\backslash l})\ v = v$.*
3. *If $\vdash v : r$ and $|v| = lw$ for some word w, then $(inj_{r\backslash l} \circ proj_{(r,l)})\ v = v$.*

MS:BUG[Come accross this issue when going back to our constructive reg-ex work] Consider $\vdash [Right\ (), Left\ a] : (a + \epsilon)^*$. However, $proj_{((a+\epsilon)^*,a)}\ [Right\ (), Left\ a]$ fails! The point is that *proj* only works correctly if applied on POSIX parse trees.

MS:Possible fixes We only ever apply *proj* on Posix parse trees.

For convenience, we write "$\vdash v : r$ is POSIX" where we mean that $\vdash v : r$ holds and v is the POSIX parse tree of r for word $|v|$.

Lemma 2 follows from the following statement.