Regular Expression Equivalence via Derivatives

Fabian Hellauer

Goal

equivalence checker for regular expressions which is

- automatic: without user interaction
- complete
- elegant, i.e. easy to prove correct

… for an Isabelle proof method

Correctness Statement

$\text{is\_bisimulation as ps} \Longrightarrow (r, s) \in \text{ps} \Longrightarrow L\ r = L\ s$

Shortcuts

Avoid having to formalize and prove correct the automata

- construction (notation!)
- determinization
- minimization

Languages

- Words are lists.
- Languages are sets of words.
- Derivative-Language:

$$D_x\ (A) := \{xs.\ x\#xs \in A\}$$

- *Interesting* languages are the infinite ones.

  →represent them by regular expressions (REs)

Bisimulations

Definition

- for all A and B

  $$A \sim B \Rightarrow [] \in A \leftrightarrow [] \in B$$

- for all A and B and x

  $$A \sim B \Rightarrow D_x\ (A) \sim D_x\ (B).$$

If "$\sim$" is a bisimulation, then $A \sim B$ implies $A = B$.

proof by list induction.

Regular Expressions

$$L(\emptyset) = \emptyset$$

$$L(\varepsilon) = \{[]\}$$

$$L(a) = \{[a]\}$$

$$L(r + s) = L(r) \cup L(s)$$

$$L(r \cdot s) = L(r)L(s)$$

$$L(r^*) = (L(r))^*$$

Derivatives of REs

- D is not computable
- use operation on REs instead: d

  goal: $L(d_a(r)) = D_a(L(r))$

  with $d :: \text{‘}a \Rightarrow \text{‘}a\ rexp \Rightarrow \text{‘}a\ rexp$ computable
- This is possible (Brzozowski 1964):

  $d_a\ (\emptyset) = \emptyset$

  $d_a\ (\varepsilon) = \emptyset$

  $d_a\ (<b>) = (\mathit{if}\ a = b\ \mathit{then}\ \varepsilon\ \mathit{else}\ \emptyset)$

  $d_a\ (r + s) = d_a\ r + d_a\ s$

Times and Star

$\mathrm{d_a(r \cdot s) =}$

$\quad$ (let $drs = d_a\ (r) \cdot s$

$\quad\quad$ in if $nullable\ r$ then $drs + d_a\ (s)$ else drs)

$\mathrm{d_a\ (r^*) = d_a\ (r) \cdot r^*}$

$\_$

$L(d_a(r)) = D_a(L(r))$ *follows by structural induction.*

Algorithm

Assume a definition

fun while where while tst stp state =

(if tst state then while tst stp (stp state) else state)

In our case, state has the type

$$('a\ rexp \times 'a\ rexp)\ list \times ('a\ rexp \times 'a\ rexp)\ list$$

Step and Test

```
fun stp where stp as (ws, ps) =
  (let ps' = hd ws # ps;
    new = [p←succs as (hd ws) . p ∉ set ps' ∪ set ws]
  in (new @ tl ws, ps'))
```

```
…where succs as (r, s) = map (λa. (nderiv a r, nderiv a s)) as
```

```
test (ws,_) ⟷
        (case ws of [] ⇒ False | (p, q)#vs ⇒ nullable p ⟷ nullable q)
```

Example

<on the board>

result

$$L\ ((\varepsilon + a \cdot b)^* \cdot (a + b)) = L\ ((a \cdot b + \varepsilon)^* \cdot (a + b))$$

In each step

- A pair from the work set is processed
- All pairs missing for the property

$$\forall a \in \text{set as. } (\text{—r, nderiv a s}) \in R)$$

are added to the work set

“as” will be the set of atoms in the expressions (this does not change during execution)

ACI

associativity, commutativity and idempotence of *+*

- the auxiliary function *norm* establishes a normal form

  ➔ ACI-equal terms are identified at that step already

- to verify this, one would have to state ACI-equivalence formally.

- the REs in the (emergent) bisimulation are kept in this normal form, as an invariant

Closure computation

- terminates if either
  - the work set is empty (bisimulation constructed)
  - a nonequivalent pair of REs is to be processed (counterexample found)

Extensions

- “⊆” can be solved easily

- extended regular expressions: to-do

→Need a computable deriv for those

<Rules>

Beyond equalities

- “$\subseteq$” can be solved easily, using

- $\neg$”$\equiv$” should really be stated differently, even though decidable

  e.g. $w \in A \setminus B$

- Relation algebras

Reflection due to Boyer and Moore

- one atom for every relation: <0>, <1>, …

- goal $(R^* \circ S^* \circ T)^* = (R \cup S \cup T)^*$

  ↝ goal (<0>*·<1>*·<2>)* = (<0> + <1> + <2>)*

Defining a usable proof method

- With *Eisbach*, one can use the usual method modifiers in a “proof method definition”:

```
method rexp = (unfold subset_eq_to_eq)?, (rule soundness, eval)+
```

- Example:

```
lemma "lang (Times (Star (Plus One AB)) A_or_B) ⊆ lang (Times (Star (Plus AB One)) A_or_B)“
  by rexp
```

References

- J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, Oct. 1964.
- A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. J. Automated Reasoning, 49:95–106, 2012. published online March 2011. http://www21.in.tum.de/~nipkow/pubs/jar12.html.
- Robert S. Boyer and J Strother Moore. Metafunctions: proving them correct and using them efficiently as new proof procedures. In R. Boyer and J Moore, editors, *The Correctness Problem in Computer Science*, pages 103–184. Academic Press, 1981