Coercions and substitutions

Fritz Henglein

DIKU, University of Copenhagen

2nd International Summer School on Advances in Programming Languages
Edinburgh, Scotland

2014-08-22

Recall: Regular expressions as types

- Language of expressions $\mathrm{Reg}_{\mathcal{A}}$:

$$E, F ::= 0 \mid 1 \mid a \mid E|F \mid EF \mid E^*$$

- Type interpretation $\mathcal{T}[\![E]\!]$:

$$\begin{aligned}
\mathcal{T}[\![0]\!] &= \emptyset \\
\mathcal{T}[\![1]\!] &= \{()\} \\
\mathcal{T}[\![a]\!] &= \{a\} \\
\mathcal{T}[\![E|F]\!] &= \mathcal{T}[\![E]\!] + \mathcal{T}[\![F]\!] \\
\mathcal{T}[\![EF]\!] &= \mathcal{T}[\![E]\!] \times \mathcal{T}[\![F]\!] \\
\mathcal{T}[\![E^*]\!] &= \mathcal{T}[\![E]\!] \ \texttt{list}
\end{aligned}$$

where

$$\begin{aligned}
S + T &= \{\mathrm{inl}\, v \mid v \in S\} \cup \{\mathrm{inr}\, w \mid w \in T\} \\
S \ \texttt{list} &= \{[v_1, \ldots, v_n] \mid v_i \in S\}
\end{aligned}$$

Regular expressions as languages

Language interpretation $\mathcal{L}[\![E]\!]$:

$$\begin{aligned}
\mathcal{L}[\![E|F]\!] &= \mathcal{L}[\![E]\!] \cup \mathcal{L}[\![F]\!] \\
\mathcal{L}[\![EF]\!] &= \mathcal{L}[\![E]\!]\,\mathcal{L}[\![F]\!] \\
\mathcal{L}[\![E^*]\!] &= \mathcal{L}[\![E]\!]^*
\end{aligned}$$

$$\begin{aligned}
S\ T &= \{st \mid s \in S \wedge t \in T\} \\
S^* &= \{s_1 s_2 \ldots s_n \mid s_i \in S\}
\end{aligned}$$

Values = parse trees = membership proofs

Views of $\mathcal{T}[\![E]\!]$:

- The values of E read as a *type* built from singletons, sum, product, list
- The parse trees of all the strings matching E
- The certificates (proofs) of membership for strings in $\mathcal{L}[\![E]\!]$.

Proposition

$\mathcal{L}[\![E]\!] = \{\mathrm{flat}(v) \mid v \in \mathcal{T}[\![E]\!]\}$

Regular Expression Containment

Definition (Language containment)

$\models E \leq F$ if $\mathcal{L}[\![E]\!] \subseteq \mathcal{L}[\![F]\!]$.

Write $\models E = F$ iff $\models E \leq F$ and $\models F \leq E$.

Definition (Coercion)

$\models f : E \leq F$ if f is function from $\mathcal{T}[\![E]\!]$ to $\mathcal{T}[\![F]\!]$ such that $\mathrm{flat}(f(v)) = \mathrm{flat}(v)$.

Theorem

$\models E \leq F$ if and only if there exists f such that $\models f : E \leq F$.

Example: Language containment by FP

- Conway's denesting rule: $\models (E|F)^* = E^*(FE^*)^*$ for all E, F.
- Proof by functional programming:
  Find $f : (\text{'a} + \text{'b})\ \text{list} \to \text{'a list} \times (\text{'b} \times \text{'a list})\ \text{list}$
  such that f does not discard, duplicate or reorder its input

```
f([]) = ([], [])
f(inl u::zs) =
    let (xs,ys) = f (zs) in (u::xs,ys)
f(inr v::zs) =
    let (xs,ys) = f (zs) in ([],(v,xs)::ys)
```

- f terminates since it is called recursively with smaller sized arguments
- f is string-preserving
- f is polymorphic, so works for all E, F. (Indeed E, F need not even be regular.)
- Therefore: $\models (E|F)^* \leq E^*(FE^*)^*$ for all E, F.
- Reverse direction similar.

Synthesis problem

- Need only *existence* of a? Decision problem, classical automata and formal language theory.
- Need to *construct* a coercion? Synthesis problem. Which coercion, how to construct and represent it?

Challenges:

- Different coercions, extensionally—choose which one? E.g. $\mathrm{tagL} : a \leq a + a$ or $\mathrm{tagR} : a \leq a + a$?
- Different coercions, intensionally—want *efficient* ones. E.g., $\mathrm{id} : a^* \leq a^*$ better than $\mathtt{map}[\mathrm{id}] : a^* \leq a^*$.
- How to design "sublanguage" (DSL) of well-typed functions that are
  - guaranteed to be coercions (soundness);
  - contain at least one coercion for every valid containment (completeness);
  - can be searched practically efficiently (not too large);
  - contain efficient coercions (not too small).

Idea: Constructive interpretation of axiomatization of containment

Axiomatization: Weak containment

shuffle : $E + (F + G) = (E + F) + G$

retag : $E + F = F + E$

untagL : $0 + F = F$

untag : $E + E \leq E$

tagL : $E \leq E + F$

assoc : $E \times (F \times G) = (E \times F) \times G$

swap : $E \times 1 = 1 \times E$

proj : $1 \times E = E$

abortR : $E \times 0 = 0$

abortL : $0 \times E = 0$

distL : $E \times (F + G) = (E \times F) + (E \times G)$

distR : $(E + F) \times G = (E \times G) + (F \times G)$

wrap : $1 + E \times E^* = E^*$

id : $E = E$

$$\frac{c : E \leq E' \qquad d : E' \leq E''}{c; d : E \leq E''}$$

$$\frac{c : E \leq E' \qquad d : F \leq F'}{c + d : E + F \leq E' + F'}$$

$$\frac{c : E \leq E' \qquad d : F \leq F'}{c \times d : E \times F \leq E' \times F'}$$

Double reading:

- Names of axioms and rules for representing derivations as terms.
- Base functions and combinators for building coercions

Computational interpretation of coercions

$$
\begin{array}{lcl}
\mathrm{shuffle}(\mathrm{inl}\, v) & = & \mathrm{inl}\,(\mathrm{inl}\, v) \\
\mathrm{shuffle}(\mathrm{inr}\,(\mathrm{inl}\, v)) & = & \mathrm{inl}\,(\mathrm{inr}\, v) \\
\mathrm{shuffle}(\mathrm{inr}\,(\mathrm{inr}\, v)) & = & \mathrm{inr}\, v \\
\mathrm{shuffle}^{-1}(\mathrm{inl}\,(\mathrm{inl}\, v)) & = & \mathrm{inl}\, v \\
\mathrm{shuffle}^{-1}(\mathrm{inl}\,(\mathrm{inr}\, v)) & = & \mathrm{inr}\,(\mathrm{inl}\, v) \\
\mathrm{shuffle}^{-1}(\mathrm{inr}\, v) & = & \mathrm{inr}\,(\mathrm{inr}\, v) \\
\mathrm{retag}(\mathrm{inl}\, v) & = & \mathrm{inr}\, v \\
\mathrm{retag}(\mathrm{inr}\, v) & = & \mathrm{inl}\, v \\
\mathrm{retag}^{-1} & = & \mathrm{retag} \\
\mathrm{untagL}\,(\mathrm{inr}\, v) & = & v \\
\mathrm{untag}\,(\mathrm{inl}\, v) & = & v \\
\mathrm{untag}\,(\mathrm{inr}\, v) & = & v \\
\mathrm{tagL}\,(v) & = & \mathrm{inl}\, v \\
\mathrm{assoc}(v, (w, x)) & = & ((v, w), x) \\
\mathrm{assoc}^{-1}((v, w), x) & = & (v, (w, x))
\end{array}
$$

Computational interpretation of coercions (2)

$$
\begin{array}{lcl}
\mathrm{swap}(v, ()) & = & ((), v) \\
\mathrm{swap}^{-1}((), v) & = & (v, ()) \\
\mathrm{proj}((), w) & = & w \\
\mathrm{proj}^{-1}(w) & = & ((), w) \\
\mathrm{distL}(v, \mathrm{inl}\, w) & = & \mathrm{inl}\,(v, w) \\
\mathrm{distL}(v, \mathrm{inr}\, x) & = & \mathrm{inr}\,(v, x) \\
\mathrm{distL}^{-1}(\mathrm{inl}\,(v, w)) & = & (v, \mathrm{inl}\, w) \\
\mathrm{distL}^{-1}(\mathrm{inr}\,(v, x)) & = & (v, \mathrm{inr}\, x) \\
\mathrm{distR}(\mathrm{inl}\, v, w) & = & \mathrm{inl}\,(v, w) \\
\mathrm{distR}(\mathrm{inr}\, v, x) & = & \mathrm{inr}\,(v, x) \\
\mathrm{distR}^{-1}(\mathrm{inl}\,(v, w)) & = & (\mathrm{inl}\, v, w) \\
\mathrm{distR}^{-1}(\mathrm{inr}\,(v, x)) & = & (\mathrm{inr}\, v, x) \\
\mathrm{wrap}\,(v) & = & \mathrm{fold}\, v \\
\mathrm{wrap}^{-1}(v) & = & \mathrm{fold}^{-1}\, v
\end{array}
$$

Computational interpretation of coercions (3)

Type constructors as *functors*:

$$\begin{array}{lcl}
\mathrm{id}(v) & = & v \\
\mathrm{id}^{-1} & = & \mathrm{id} \\
(c; d)(v) & = & d(c(v)) \\
(c + d)(\mathrm{inl}\, v) & = & \mathrm{inl}\,(c(v)) \\
(c + d)(\mathrm{inr}\, w) & = & \mathrm{inr}\,(d(w)) \\
(c \times d)(v, w) & = & (c(v), d(w))
\end{array}$$

Kozen Axiomatization (1994)

$$\begin{array}{lcrcl}
\text{shuffle} & : & E + (F + G) & = & (E + F) + G \\
\text{retag} & : & E + F & = & F + E \\
\text{untagL} & : & 0 + F & = & F \\
\text{untag} & : & E + E & \leq & E \\
\text{tagL} & : & E & \leq & E + F \\
\text{assoc} & : & E \times (F \times G) & = & (E \times F) \times G \\
\text{swap} & : & E \times 1 & = & 1 \times E \\
\text{proj} & : & 1 \times E & = & E \\
\text{abortR} & : & E \times 0 & = & 0 \\
\text{abortL} & : & 0 \times E & = & 0 \\
\text{distL} & : & E \times (F + G) & = & (E \times F) + (E \times G) \\
\text{distR} & : & (E + F) \times G & = & (E \times G) + (F \times G) \\
\text{wrap} & : & 1 + E \times E^* & = & E^* \\
\text{id} & : & E & = & E
\end{array}$$

$$\frac{c : E \leq E' \qquad d : E' \leq E''}{c; d : E \leq E''}$$

$$\frac{c : E \leq E' \qquad d : F \leq F'}{c + d : E + F \leq E' + F'}$$

$$\frac{c : E \leq E' \qquad d : F \leq F'}{c \times d : E \times F \leq E' \times F'}$$

$$\frac{c : E \times F \leq F}{\text{foldr[c]} : E^* \times F \leq F}$$

$$\frac{c : E \times F \leq E}{\text{foldl[c]} : E \times F^* \leq E}$$

Example proofs

$$\mathrm{cons} = \ldots : a \times a^* \leq a^*$$

$$\mathrm{foldl[foldr[cons]]} : a^* \times a^{**} \leq a*$$

Short proof. But: $\mathrm{foldl[foldr[cons]]}$ runs in quadratic time!
Does Kozen's axiomatization contain a "faster" proof?

Henglein-Nielsen axiomatization (2011)

- Weak axiomatiation + 1 rule:
  Safe coinduction (= terminating recursion).
- Contains all proofs of Salomaa, Kozen, Grabmayer.
- Numerous containment proofs that yield the *identity* (noop) on bit-coded parse trees or a *finite state transducer* (streaming linear-time, with constant-sized buffer).
  - How to efficiently synthesize these? (Future work)

A (Somewhat) Realistic Scenario

Fix malformed CSV data:

`12,0,11,2,13,1`

↓

`12,0;11,2;13,1`

RE

$$r = \texttt{[0-9]+,[0-9]+}$$
$$t = \texttt{(}r\texttt{,)*}r$$

$L(t)$ contains all

$$w_0\texttt{,}w_1\texttt{,}\cdots\texttt{,}w_{n-1}\texttt{,}w_n$$

where $w_i \in L(r)$.

A PCRE attempt

Perl Compliant Regular Expressions (PCRE): Only one match under Kleene star.

$$\texttt{s/(}r\texttt{,)*(}r\texttt{)/???/}$$

A PCRE attempt

Perl Compliant Regular Expressions (PCRE): Only one match under Kleene star.

$$\texttt{s/}\underbrace{\texttt{(}r\texttt{,)}}_{1}\texttt{*}\underbrace{\texttt{(}r\texttt{)}}_{2}\texttt{/???/}$$

A PCRE attempt

Perl Compliant Regular Expressions (PCRE): Only one match under Kleene star.

$$\texttt{s/}\underbrace{\texttt{(}r\texttt{,)}}_{1}\texttt{*}\underbrace{\texttt{(}r\texttt{)}}_{2}\texttt{/???/}$$

$$\underbrace{w_0\,,w_1\,,\,\cdots\,,}_{\text{lost!}}\underbrace{w_{n-1}\,,}_{1}\underbrace{\;w_n\;}_{2}$$

Using PCRE

Iteration needs to be hand-coded.
In Python:

```
e = '([0-9]+,[0-9]+)|(,)'
for m in re.finditer(e, text):
  if m.group(1):
    out += m.group(1)
  else:
    out += ";"
```

Substitutions, functorially

`(r,)*r`

Substitutions, functorially

$$\texttt{(r,)*r} \sim \mathsf{List}(r \times \{,\}) \times r$$

Substitutions, functorially

$$\texttt{(r,)*r} \sim \mathsf{List}(r \times \{,\}) \times r$$

Matches are *values*:

$$([(w_1, \underline{,}), \ldots, (w_{n-1}, \underline{,})], w_n)$$

Substitutions, functorially

$$\begin{tikzcd}
\mathtt{(r,)*r} \sim \mathsf{List}(r \times \{,\}) \times r \arrow[d, "(\mathtt{id} \times \mathtt{semic})^{\star} \times \mathtt{id}"] \\
\mathsf{List}(r \times \{;\}) \times r
\end{tikzcd}$$

Matches are *values*:

$$([(w_1, \underline{,}), \ldots, (w_{n-1}, \underline{,})], w_n)$$

Substitutions, functorially

$$\begin{tikzcd}
\texttt{(r,)*r} \sim \mathsf{List}(r \times \{,\}) \times r \arrow[d, "(\mathtt{id} \times \mathtt{semic})^{\star} \times \mathtt{id}"] \\
\mathsf{List}(r \times \{;\}) \times r
\end{tikzcd}$$

Matches are *values*:

$$([(w_1, \underline{;}), \ldots, (w_{n-1}, \underline{;})], w_n)$$

Nested Repetition

$$\mathsf{List}\big(\mathsf{List}(r \times \{\texttt{,}\}) \times r \times \{\texttt{\textbackslash n}\}\big)$$

Nested Repetition

$$\begin{tikzcd}
\mathsf{List}\big(\mathsf{List}(r \times \{\texttt{,}\}) \times r \times \{\texttt{\textbackslash n}\}\big) \arrow[d, "\phi"] \\
\mathsf{List}\big(\mathsf{List}(r \times \{\texttt{;}\}) \times r \times \{\texttt{\textbackslash n}\}\big)
\end{tikzcd}$$

$$\phi = ((\texttt{id} \times \texttt{semic})^\star \times \texttt{id} \times \texttt{id})^\star$$

Using PCRE

Nested iteration needs to be hand-coded.

```
e = '([0-9]+,[0-9]+)|(,)'
for l in text.split('\n'):
  for m in re.finditer(e, l):
    ...
  out += "\n"
```

Summary

- Coercion = Transformation that changes RE but retains string
    - Proofs of containment induce string representation transformations
- Substitution = Structural transformation that changes strings
    - Includes *projections*, discarding parts of the input
- Synthesizing *efficient* coercions important (future work)
- Functorial transformations to express substitutionsUseful for compact RE-specific representations of strings