crsx.sourceforge.net
An Open Source Platform for Experiments with Higher Order Rewriting

Kristoffer H. Rose

http://www.research.ibm.com/people/k/krisrose

mailto:krisrose@us.ibm.com

IBM Thomas J. Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598, USA

Workshop on Higher Order Rewriting
June 25, 2007, Paris, France

With apologies for my absence!

crsx.sourceforge.net
An Open Source Platform for Experiments with Higher Order Rewriting

Kristoffer H. Rose

http://www.research.ibm.com/people/k/krisrose

mailto:krisrose@us.ibm.com

IBM Thomas J. Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598, USA

Workshop on Higher Order Rewriting
June 25, 2007, Paris, France

With apologies for my absence!

What?

`crsx.sourceforge.net`

1. implements CRS in Java
2. is open source (CPL)

Why?

`crsx.sourceforge.net` hopes to

1. provide a generic higher order rewrite engine that
2. is easy to embed in other projects such as compiler optimizers,
3. is simple to extend with experimental features, and
4. runs on a universally available open source platform.

. . . so far – depends on who joins!

Why?

`crsx.sourceforge.net` hopes to

1. provide a generic higher order rewrite engine that
2. is easy to embed in other projects such as compiler optimizers,
3. is simple to extend with experimental features, and
4. runs on a universally available open source platform.

. . . so far – depends on who joins!

How?

`crsx.sourceforge.net` needs everyone's specialized help...

Plan

What I did so far with `crsx.sourceforge.net`:

- CRS for everything (the CRS ⟅*tricks*⟆).
- Retrofitting CRS+⟅*tricks*⟆ onto Java terms.
- XQuery compilation examples.

Rewriting as usual...

Definition (CRS/Combinatory Reduction System)

Terms restrict binders to occur in constructions:

$$t ::= v \mid f(b_1, \ldots, b_n) \mid z(t_1, \ldots, t_n)$$
$$b ::= v \,.\, b \mid t$$

Rules $t_L \to t_R$ (as usual) define *rewrite relation* $\underset{R}{\to}$ of all pairs $C[\sigma(t_L)] \underset{R}{\to} C[\sigma(t_R)]$ for some context $C[]$ and valuation map $\sigma \colon Z \rightharpoonup (V^* \times T)_\perp$ where each $\sigma(z) = \langle\langle v_1, \ldots, v_n\rangle, t\rangle$ with distinct $v_1 \ldots v_n$ and $fv(t) \subseteq \{v_1, \ldots, v_n\}$ means that $\sigma(t)$ is the homomorphic extension to terms of the substitution

$$\sigma(z(t_1, \ldots, t_n)) = t[v_1 := \sigma(t_1), \ldots, v_n := \sigma(t_n)]$$

CRS ⟅tricks⟆

CRS can encode many things by term transformation, such as

1. annotations,
2. context tricks for propagation, and
3. static reduction;

as follows. . .

Annotations

Definition (Annotated CRS)

Add annotation layer for k annotations around original unannotated terms:

$$t ::= v \mid !(\, f(b_1, \ldots, b_n)\, , a_1, \ldots, a_k) \mid z(t_1..t_n)$$
$$b ::= !(\, v \,.\, b \,, a_1, \ldots, a_k) \mid t$$
$$a ::= ? \mid \ldots$$

Variables do not have properties, only *binders*.

Theorem

For every CRS there is an equivalent k-annotated CRS.

Easy proof by populating with dummy annotations.

Annotations

Definition (Annotated CRS)

Add annotation layer for k annotations around original unannotated terms:

$$
\begin{aligned}
t &::= v \mid !(\, f(b_1, \ldots, b_n)\,, a_1, \ldots, a_k) \mid z(t_1..t_n) \\
b &::= !(\, v \,.\, b\,, a_1, \ldots, a_k) \mid t \\
a &::= ? \mid \ldots
\end{aligned}
$$

Variables do not have properties, only *binders*.

Theorem

For every CRS there is an equivalent k-annotated CRS.

Easy proof by populating with dummy annotations.

Annotations

Definition (Annotated CRS)

Add annotation layer for k annotations around original unannotated terms:

$$t ::= v \mid !(\, f(b_1, \ldots, b_n)\, , a_1, \ldots, a_k) \mid z(t_1..t_n)$$
$$b ::= !(\, v \,.\, b\, , a_1, \ldots, a_k) \mid t$$
$$a ::= ? \mid \ldots$$

Variables do not have properties, only *binders*.

Theorem

For every CRS there is an equivalent k-annotated CRS.

Easy proof by populating with dummy annotations.

Annotations

Definition (Annotated CRS)

Add annotation layer for k annotations around original unannotated terms:

$$
\begin{aligned}
t &::= v \mid !(\, f(b_1, \ldots, b_n)\, , a_1, \ldots, a_k) \mid z(t_1..t_n) \\
b &::= !(\, v \,.\, b \,, a_1, \ldots, a_k) \mid t \\
a &::= \,? \mid \ldots
\end{aligned}
$$

Variables do not have properties, only *binders*.

Theorem

For every CRS there is an equivalent k-annotated CRS.

Easy proof by populating with dummy annotations.

Example XQuery annotation rules

```
R:without(
 R:alias(fs:distinct-doc-order-or-atomic-sequence(R:_()), R:Expr()),
 'ord')
→
R:with(R:Expr(), 'ord')

R:without(
 R:alias(fs:distinct-doc-order-or-atomic-sequence(R:_()), R:Expr()),
 'nodup')
→
R:with(R:Expr(), 'nodup')

fs:distinct-doc-order(R:with(R:with(R:Seq(), 'ord'), 'nodup'))
→
R:Seq()
```

Example XQuery type annotation rules

Types are seen as an annotation.

```
let $v := R:type(R:Expr1() instance of R:Type1)
return R:Expr2($v)
→
let $v as R:Type1 := R:Expr1() return R:Expr2($v)

R:without(R:alias(
  let $v as R:Type1 := R:Expr1()
  return R:type(R:Expr2($v) instance of R:Type2),
 R:Expr()), 'type')
→
R:type(R:Expr() instance of R:Type2)
```

Example XQuery type annotation rules

Types are seen as an annotation.

```
let $v := R:type(R:Expr1() instance of R:Type1)
return R:Expr2($v)
→
let $v as R:Type1 := R:Expr1() return R:Expr2($v)

R:without(R:alias(
  let $v as R:Type1 := R:Expr1()
  return R:type(R:Expr2($v) instance of R:Type2),
 R:Expr()), 'type')
→
R:type(R:Expr() instance of R:Type2)
```

Hack alert I

Free variables are allowed in patterns!

→ realized by considering free variables in patterns as metaapplication patterns of a special sort that only match variables. . .

Hack alert II

The annotation mechanism is not integrated with binding!

→ implementation cheats by allowing annotations on variable binders to be matched against variable *occurrences*. . .

Are these safe?

Hack alert I

Free variables are allowed in patterns!

→ realized by considering free variables in patterns as metaapplication patterns of a special sort that only match variables. . .

Hack alert II

The annotation mechanism is not integrated with binding!

→ implementation cheats by allowing annotations on variable binders to be matched against variable *occurrences*. . .

Are these safe?

Hack alert I

Free variables are allowed in patterns!

$\rightarrow$ realized by considering free variables in patterns as metaapplication patterns of a special sort that only match variables. . .

Hack alert II

The annotation mechanism is not integrated with binding!

$\rightarrow$ implementation cheats by allowing annotations on variable binders to be matched against variable *occurrences*. . .

Are these safe?

Hack alert I

Free variables are allowed in patterns!

→ realized by considering free variables in patterns as metaapplication patterns of a special sort that only match variables. . .

Hack alert II

The annotation mechanism is not integrated with binding!

→ implementation cheats by allowing annotations on variable binders to be matched against variable *occurrences*. . .

Are these safe?

Hack alert I

Free variables are allowed in patterns!

$\rightarrow$ realized by considering free variables in patterns as metaapplication patterns of a special sort that only match variables...

Hack alert II

The annotation mechanism is not integrated with binding!

$\rightarrow$ implementation cheats by allowing annotations on variable binders to be matched against variable *occurrences*...

Are these safe?

Propagation

Annotations (with variable hacks) can be used to implement *attribute grammars* and *deterministic inference rules*:

$$\frac{a_1 \vdash b_1 \twoheadrightarrow c_1 \quad \cdots \quad a_n \vdash b_n \twoheadrightarrow c_n}{a \vdash B[b_1, \ldots, b_n] \twoheadrightarrow c}$$

encoded by *initialization*

$$B[b_1, \ldots, b_n]^{(\textit{context}:a,\textit{state}:\bullet)} \to B[b_1^{(\textit{context}:a_1)}, \ldots, b_n]^{(\textit{context}:a,\textit{state}:1)}$$

transfer for $i \in 1..n-1$:

$$B[c_1^{(\textit{state}:\checkmark)}, \ldots, c_i^{(\textit{state}:\checkmark)}, b_{i+1}, \ldots, b_n]^{(\textit{context}:a,\textit{state}:i)}$$
$$\to B[c_1^{(\textit{state}:\checkmark)}, \ldots, c_i^{(\textit{state}:\checkmark)}, b_{i+1}^{(\textit{context}:a_{i+1})}, \ldots, b_n]^{(\textit{context}:a,\textit{state}:i+1)}$$

conclusion

$$B[c_1^{(\textit{state}:\checkmark)}, \ldots, c_n^{(\textit{state}:\checkmark)}]^{(\textit{context}:a,\textit{state}:n)} \to c^{(\textit{state}:\checkmark)}$$

Virtualization

Assume terms T, variables V, and metavariables Z, describe CRS as collection of operations:

$$
\begin{array}{rll}
\#\colon \mathrm{T} \to \mathrm{N} & \text{with N the natural numbers from 0} & \text{(arity)} \\
\mathrm{v}\colon \mathrm{T} \to V_\perp & & \text{(variable occurrence check)} \\
\mathrm{z}\colon \mathrm{T} \to Z_\perp & & \text{(metavariable check)} \\
\mathrm{b}\colon \mathrm{T} \times \mathrm{N} \to (V^*)_\perp & & \text{(binders)} \\
\mathrm{s}\colon \mathrm{T} \times \mathrm{N} \to \mathrm{T}_\perp & & \text{(subterms)} \\
\mathrm{m}\colon \mathrm{T} \times \mathrm{T} \times \Sigma_\perp \to \Sigma_\perp & & \text{(match)} \\
\mathrm{cc}\colon \mathrm{T} \times \mathrm{B}^* \to \mathrm{T} & \text{where } \mathrm{B} = V^* \times \mathrm{T} & \text{(copy constructor)} \\
\mathrm{cv}\colon V \to \mathrm{T} & & \text{(copy variable occurrence)}
\end{array}
$$

with an appropriate redefinition of rewriting. . .

Hack alert III

All rewrites are destructive updates.

$\longrightarrow$ contexts are preserved

Hack alert III

All rewrites are destructive updates.

$\longrightarrow$ contexts are preserved

Realization in Java

```
interface CRSTerm {
 enum CRSKind {CONSTRUCTOR, VARIABLE_OCCURRENCE, META_APPLICAT
 public CRSKind crsKind();

 int crsArity(); // #
 CRSVariable crsVariable(); // v
 String crsMetaVariable(); // z
 CRSVariable[] crsBinders(int i); // b
 CRSTerm crsSub(int i); // s
 boolean crsPreMatch(CRSTerm other, CRS crs); // m (1 of 2)
 boolean crsPostMatch(CRSTerm other, CRSMatching m); // m (2 of 2)
 CRSTerm crsCopyConstructor(CRSVariable[][] bs, CRSTerm[] ts); // cc
 CRSTerm crsCopyVariableOccurrence(CRSVariable v); // cv

 void crsReplaceSub(int i, CRSTerm t); C[] (1 of 2)

 CRSTerm crsMetaApplicationSubstitution(CRSValuation sigma, int sequenceno,
            CRSRenaming renaming, CRSTerm copy); C[] (1 of 2)
}
```

XQuery encoding

1. Original abstract syntax terms extended to implement CRSTerm.
   - $\longrightarrow$ can also use delegation.
2. Types and analysis properties are encoded with annotations as discussed previously.
3. Type rules and sorting elimination rules are encoded using the inference system encoding.
4. With the current CRSX interpreter it is slow but not unreasonably so (compiles about 1000 queries/minute on laptop).

Achieved

1. Prototype quality CRS engine over abstract Java terms:
    - Untyped.
    - Interpreted.
2. Reasonable fixed normalization heuristics.
3. Bag of tricks for analysis and structured rewriting.
4. Proven with XQuery analysis and optimization.

Future work

1. Compile CRS rules directly into Java.
2. Pluggable (compiled) rewrite strategies.
3. Types?
4. Termination (and other CRS analysis)?

How?

`crsx.sourceforge.net` needs everyone's specialized help. . .

The End