Introduction

CRS as generic rewrite formalism
Virtualization for Java
Conclusion

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!
1. Introduction

2. CRS as generic rewrite formalism
   - Definition
   - Tricks

3. Virtualization for Java

4. Conclusion
What?

crsx.sourceforge.net

1 implements CRS in Java
2 is open source (CPL)
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!
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!
crsx.sourceforge.net needs everyone’s specialized help...
What I did so far with crsx.sourceforge.net:

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

2 CRS as generic rewrite formalism
   - Definition
   - Tricks

3 Virtualization for Java

4 Conclusion
Rewriting as usual...

**Definition (CRS/Combinatory Reduction System)**

Terms restrict binders to occur in constructions:

\[ t ::= v \mid f(b_1, \ldots, b_n) \mid z(t_1, \ldots, t_n) \]
\[ b ::= v \cdot b \mid t \]

Rules \( t_L \rightarrow t_R \) (as usual) define rewrite relation \( \overset{\rightarrow}{R} \) of all pairs \( C[\sigma(t_L)] \overset{\rightarrow}{R} C[\sigma(t_R)] \) for some context \( C[] \) and valuation map \( \sigma: Z \rightarrow (V^* \times T)_{\bot} \) where each \( \sigma(z) = \langle \langle v_1, \ldots, v_n \rangle, t \rangle \) with distinct \( v_1 \ldots v_n \) and \( \text{fv}(t) \subseteq \{v_1, \ldots, v_n\} \) 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 can encode many things by term transformation, such as

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

as follows...
Definition (Annotated CRS)

Add annotation layer for $k$ annotations around original unannotated terms:

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

$$
b ::= !(v \cdot b, a_1, \ldots, a_k) \mid t
$$

$$
a ::= ? \mid \ldots
$$

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:

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

$b ::= !(v \cdot b, a_1, \ldots, a_k) \mid t$

$a ::= ? \mid \ldots$

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:

\[
\begin{align*}
\text{t} &::= \text{v} \mid !\left( f(b_1, \ldots, b_n), a_1, \ldots, a_k \right) \mid z(t_1 \ldots t_n) \\
\text{b} &::= \left( v \cdot b, a_1, \ldots, a_k \right) \mid t \\
\text{a} &::= ? \mid \ldots
\end{align*}
\]

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:

\[
\begin{align*}
t &::= v \mid \bang (f(b_1, \ldots, b_n), a_1, \ldots, a_k) \mid z(t_1..t_n) \\
b &::= \bang (v \cdot b, a_1, \ldots, a_k) \mid t \\
a &::= ? \mid \ldots
\end{align*}
\]

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

\[ R:without(\]
  \[ R:alias(fs:distinct-doc-order-or-atomic-sequence(R:_(), R:Expr()), 'ord') \]
\[ \rightarrow \]
\[ R:with(R:Expr(), 'ord') \]
\[ R:without(\]
  \[ R:alias(fs:distinct-doc-order-or-atomic-sequence(R:_(), R:Expr()), 'nodup') \]
\[ \rightarrow \]
\[ R:with(R:Expr(), 'nodup') \]
\[ \]
\[ fs:distinct-doc-order(R:with(R:with(R:Seq(), 'ord'), 'nodup')) \]
\[ \rightarrow \]
\[ R:Seq() \]
Example XQuery type annotation rules

Types are seen as an annotation.

```xquery
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)
```
Types are seen as an annotation.

\[
\text{let } \$v := \text{R:type(}R:\text{Expr1()} \text{ instance of R:Type1)} \\
\text{return } R:\text{Expr2(}\$v) \\
\rightarrow \\
\text{let } \$v \text{ as R:Type1 := R:Expr1()} \text{ return R:Expr2(}\$v)
\]

\[
\text{R:without(R:alias(} \\
\text{let } \$v \text{ as R:Type1 := R:Expr1()} \\
\text{return R:type(R:Expr2(}\$v \text{ instance of R:Type2)}, \\
R:\text{Expr()}), 'type')} \\
\rightarrow \\
\text{R:type(R:Expr()} \text{ instance of R:Type2)}
\]
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!

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

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

encoded by initialization

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

transfer for \( i \in 1..n - 1 \):

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

conclusion

\[ B[c_1^{(\text{state}: \checkmark)}, \ldots, c_n^{(\text{state}: \checkmark)}]^{(\text{context}: a, \text{state}: n)} \rightarrow c^{(\text{state}: \checkmark)} \]
1 Introduction

2 CRS as generic rewrite formalism
   - Definition
   - Tricks

3 Virtualization for Java

4 Conclusion
Virtualization

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

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

with an appropriate redefinition of rewriting...
All rewrites are destructive updates.

→ contexts are preserved
Hack alert III

All rewrites are destructive updates.

→ contexts are preserved
interface CRSTerm {
    enum CRSKind {CONSTRUCTOR, VARIABLE_OCCURRENCE, META_APPLICATION);
    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.
   → 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).
Introduction

CRS as generic rewrite formalism

- Definition
- Tricks

Virtualization for Java

Conclusion
Achieved

1. Prototype quality CRS engine over abstract Java terms:
   - Untyped.
   - Interpreted.

2. Reasonable fixed normalization heuristics.


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