A Unification Algorithm to Compute Overlaps in a Call-by-Need Lambda-Calculus with Variable-Binding Chains

Conrad Rau and Manfred Schmidt-Schauß

Goethe-University, Frankfurt am Main, Germany

UNIF 2011, Wroclaw, Poland
Introduction

Motivation: **Analysis of programming language semantics**

Programming language:
Core language, model by an extended $\lambda$-calculus

Semantics: Contextual equivalence of programs
Correctness of program transformations: $P_1 \xrightarrow{T} P_2$?

Step in correctness proof:
Determination of overlaps of reduction rules of the $\lambda$-calculus

**Compute overlaps of reduction rules** in $\lambda$-calculi via **unification**

Overview of the Talk
Overlap computation for a specific $\lambda$-calculus $LR$
Own and Related Work

- R., Schmidt-Schauß, 2010
  Overlaps in a simple extended $\lambda$-calculus

- Schmidt-Schauß, Schütz, Sabel, 2008
  Extended $\lambda$-calculus LR and correctness of program transformations

- Dantsin, Voronkov, 1999
  Unification of sets and multi-sets

- Comon, 1998
  Unification with context variables

- Salzer, 1992
  Unification of (infinite) sets of terms (schematization)
The $\lambda$-calculus $LR$

Core language of pure Haskell

Syntax of LR expressions

\[ s, s_1, \ldots, s_n \in LR ::= \]
\[ x \mid (\lambda x. s) \mid (s_1 \ s_2) \mid \text{letrec } x_1 = s_1, \ldots, x_n = s_n \text{ in } s \]
\[ (\text{seq } s_1 \ s_2) \mid (c \ s_1 \ldots s_n) \mid (\text{case}_T \ s \ \text{Alt}_1 \ldots \text{Alt}_{|T|}) \]
\[ \text{Alt} ::= (\text{Pat} \rightarrow \ s) \]
\[ \text{Pat} ::= (c \ x_1 \ldots x_n) \]

Syntactic restrictions and side conditions for expressions:

- Expressions satisfy the DVC
- In the letrec-environment \( \{x_1 = s_1, \ldots, x_n = s_n\} \) the bindings are commutative
- ...
Operational Semantics of LR: Reduction Rules

Transformations (excerpt)

(lbeta) \((\lambda x.s) r) \rightarrow (\text{letrec } x = r \text{ in } s)\)
(seq-in) \((\text{letrec } x_1 = v, \{x_{i+1} = x_i\}_{i=1}^m, E \text{ in } C[(\text{seq } x_m t)]) \rightarrow (\text{letrec } x_1 = v, \{x_{i+1} = x_i\}_{i=1}^m, E \text{ in } C[t])\)
if \(v\) is a constructor application

Normal Order Reduction (Call-by-Need) \(\stackrel{\text{no}}{\rightarrow}\) (excerpt)

(no, lbeta) \(A[(\lambda x.s) r] \rightarrow A[\text{letrec } x = r \text{ in } s]\)
(no, cp-e)

\begin{align*}
\text{letrec } x_1 &= v, \{x_{i+1} = x_i\}_{i=1}^m, y_1 = A_1[x_m], \{y_{i+1} = A_{i+1}[y_i]\}_{i=1}^n, E \text{ in } A[y_n] \\
\rightarrow \text{letrec } x_1 &= v, \{x_{i+1} = x_i\}_{i=1}^m, y_1 = A_1[v], \{y_{i+1} = A_{i+1}[y_i]\}_{i=1}^n, E \text{ in } A[y_n]
\end{align*}
where \(v\) is an abstraction and \(A_1\) is a non-empty context.

Rules contain special syntactic constructs:

- **Meta variables** to denote arbitrary terms
- **Context**: Expression with one hole \([\cdot]\) of different classes: \(A < S < C\)
- **Binding chains** of variable length (schemas)

\[
A \in A ::= [\cdot] \mid (A \ s) \mid (\text{case}_T A \ Alts) \mid (\text{seq} A \ s)
\]
\[
S \in S ::= [\cdot] \mid (S \ s) \mid (s \ S)
\]
\[
\vdots
\]
To show a LR transformation \( T \) as correct, compute all overlaps between \( T \) and all normal order reductions

Single overlap \( t_1 \xrightarrow{T} t \xrightarrow{no,a} t_2 \)

For transformation \( T \), no-reduction \( a \):
all overlaps are described through the solutions of
\( S[l_1] \dashv l_2 \), where
- \( l_1 \rightarrow r_1 \) is an LR-transformation
- \( l_2 \rightarrow r_2 \) is an LR-normal order reduction

All overlaps between transformations and no-reductions are described by

\[
IP := \{ \{ S[l_1] \dashv l_2 \} \mid l_1 \in lhs_T, l_2 \in lhs_{no} \}
\]

the initial LR-forking-problem
For unification we encode
\( LR \)-rules as \textbf{many sorted term} with \textbf{special construct}

If \([\cdot]\) is the translation from expressions to terms we solve
\[
\{ \{ [S[l_1]] \doteq [l_2] \} \mid l_1 \in \text{lhs}_{T}, l_2 \in \text{lhs}_{no} \}
\]
Encoding of LR-expressions as Many Sorted Terms

\[
\begin{align*}
[x] &= \text{var}(x) & \text{var} :: BV \to \text{Exp}, \\
[(\lambda x.s)] &= \text{lam}(x, [s]) & \text{lam} :: BV \times \text{Exp} \to \text{Exp} \\
[(s_1 \ s_2)] &= \text{app}([s_1], [s_2]) & \text{app} :: \text{Exp} \times \text{Exp} \to \text{Exp} \\
[\text{seq} \ s_1 \ s_2] &= \text{seq}([s_1], [s_2]) & \text{seq} :: \text{Exp} \times \text{Exp} \to \text{Exp} \\
[(c \ s_1 \ldots s_n)] &= c([s_1], \ldots, [s_n]) & c :: \text{Exp} \times \ldots \times \text{Exp} \to \text{Exp} \\
[(\text{case}_T \ s \ \text{At}_1 \ldots \text{At}_{|T|})] &= \text{case}_T([s], [\text{At}_1], \ldots, [\text{At}_{|T|}]) \\
[(\text{letrec} \ Env \ \text{in} \ s)] &= \text{let}([Env], [s]) & \text{let} :: Env \times \text{Exp} \to \text{Exp} \\
[x = s] &= \text{bind}(x, [s]) & \text{bind} :: BV \times \text{Exp} \to \text{Bind} \\
[\{\}] &= \emptyset_{env} & \emptyset_{env} :: Env \\
[\{x_1 = s_1, \ldots, x_n = s_n\}] &= \text{env}([x_1 = s_1], [\{\ldots, x_n = s_n\}]) & \text{env} :: \text{Bind} \times \text{Env} \to \text{Env}
\end{align*}
\]

Abbreviation:

\[\text{env}(b_1, \text{env}(b_2, \ldots, \text{env}(b_n, \emptyset_{env}) \ldots)) = \text{env}^*(\{b_1, \ldots, b_n\} \cup \emptyset_{env})\]
Encoding LR Reduction Rules

- **Commutativity of bindings** in letrec-environments:

\[ LC_{env} := \{ env(x, env(y, z)) = env(y, env(x, z)) \} \]

- Encoding of **special constructs** from LR-reduction rules:
  - **Meta variable** encoded as variable of appropriate sort
  - **Context** encoded as context variable of class \( A, S \) or \( C \)
  - **Binding chains**:
    - \( \{ x_{i+1} = x_i \}_{i=1}^m \) encoded as \( \text{VCh}(x_1, x_m, k) \)
    - \( \{ y_{i+1} = A_{i+1}[y_i] \}_{i=1}^n \) encoded as \( \text{NCh}(y_1, y_n, l) \)

**Example**

\[
\text{letrec } x_1 = v, y_1 = A_1[x_m], \quad \text{let}(\text{env}^* (\{\text{bind}(x_1, v), \text{bind}(y_1, A_1(\text{var}(x_m)))\}) \\
\quad \cup \text{VCh}(x_1, x_m, k) \\
\quad \cup \text{NCh}(y_1, y_n, l) \cup E), \\
\quad A(\text{var}(y_n))
\]

\[
\text{letrec } x_1 = v, y_1 = A_1[x_m], \quad \text{let}(\text{env}^* (\{\text{bind}(x_1, v), \text{bind}(y_1, A_1(\text{var}(x_m)))\}) \\
\quad \cup \text{VCh}(x_1, x_m, k) \\
\quad \cup \text{NCh}(y_1, y_n, l) \cup E), \\
\quad A(\text{var}(y_n))
\]
Solving \( IP \)

- \( IP := \{[[S(l_1)] \doteq [l_2]] \mid l_1 \in lhs_T, l_2 \in lhs_{no}\} \)

- **Non-deterministic unification algorithm** that transforms \( S; P, P \in IP \) and stops with \( \text{Fail} \) or \( S, \emptyset \)

- Properties of \( IP \):
  - Each \( P \in IP \) is **almost linear**
  - In each \( P \) there are at most 3 chains

- A solution \( \sigma \) of \( P = \{S(l_1) \doteq l_2\} \) must satisfy:
  - \( \sigma(S(l_1)) =_{LC} \sigma(l_2) \)
  - \( [[\sigma(S(l_1))]]^{-}, [[\sigma(l_2)]]^{-} \) are **admissible LR-expressions**: they satisfy the syntactic restrictions and side conditions of \( LR \)

**Example**

\[
\left[\text{let}(\text{env}^* (\{\text{bind}(x, \text{var}(y)), \text{bind}(x, \text{app}(\text{var}(x), \text{var}(x)))\} \cup \emptyset_{env}), \text{var}(x))\right]^{-} = \text{letrec } x = y, \ x = (x \ x) \ \text{in } x
\]

violates the \( DVC \)
### Standard Unification Rules

#### Standard Rules

**Solve**

\[
\frac{S; \{x \doteq t\} \uplus P}{\{x \doteq t\} \cup S; \ P}
\]

**Trivial**

\[
\frac{S; \{s \doteq s\} \uplus P}{S; \ P}
\]

**Dec**

\[
\frac{S; \{f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n)\} \uplus P}{S; \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\} \cup P}
\]

- If \(f \neq \text{env}\)

**Fail**

\[
\frac{S; \{f(\ldots) \doteq g(\ldots)\} \uplus P}{\text{Fail}}
\]

- If \(f \neq g\)
Solving Equations with Context Variables

**Solve** $X(s) \vdash f(t_1, \ldots, t_n)$

Guess the position where $s$ can occur in $f(t_1, \ldots, t_n)$ thereby take context classes into account

**Context Rules (excerpt)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
</table>
| Dec-C-App-Seq      | $S; \{X(s) \vdash f(t_1, t_2)\} \cup P; X \in \Delta$
|                    | $\{X \vdash f(X', t_2)\} \cup S; \{X'(s) \vdash t_1\} \cup P; \Delta$
|                    | $f \in \{\text{app, seq}\}, X'$ is a fresh of same class as $X$            |
| Dec-C              | $S; \{X(s) \vdash f(t_1, t_2)\} \cup P; X \in \Delta$
|                    | $\{X \vdash f(t_1, X')\} \cup S; \{X'(s) \vdash t_2\} \cup P; \Delta$
|                    | $f \in \{\text{let, app, seq}\}, cc(X) > A, X'$ is fresh, of same class as $X$ |
Solving Equations with Context Variables

Solve $X(s) \triangleright= Y(t)$: Either

holes are at a comparable position (common prefix)
holes are at an incomparable position (guess fork $f'$)

Context Rules (excerpt)

Merge-Prefix

$$S; \{X(s) \triangleright= Y(t)\} \uplus P; \ X, Y \in \Delta$$

$$\{Y \triangleright= ZY', X \triangleright= ZX'\} \cup S; \ \{X'(s) \triangleright= Y'(t)\} \cup P; \ \Delta$$

$X', Y', Z$ are fresh of the same context class as $X, Y$, and $Z$ the smaller context class of $X, Y$

Merge-Fork-A

$$S; \{X(s) \triangleright= Y(t)\} \uplus P; \ X, Y \in \Delta$$

$$1) \ \{X \triangleright= app(X', Y'(t)), Y \triangleright= app(X'(s), Y')\} \cup S; \ P; \ \Delta$$

...
Solving Multi-Set Equations

**LC Unification: Binding - Binding**

**Solve-Env**

\[
S; \{ \text{env}^* (M_1 \cup r_1) \doteq \text{env}^* (M_2 \cup r_2) \} \cup P
\]

\[
\{ r_1 \doteq \text{env}^* (M_2 \cup z_3), r_2 \doteq \text{env}^* (M_1 \cup z_3) \} \cup S; P
\]

if \( r_1, r_2 \) are variables and \( z_3 \) is a fresh variable

**Dec-Env**

\[
S; \{ \text{env}^* (M_1 \cup r_1) \doteq \text{env}^* (M_2 \cup r_2) \} \cup P
\]

\[
S; \{ t_1 \doteq t_2, \text{env}^* ((M_1 \setminus \{t_1\}) \cup r_1) \doteq \text{env}^* ((M_2 \setminus \{t_2\}) \cup r_2) \} \cup P
\]

if \( t_1 \in M_1 \) and \( t_2 \in M_2 \)

**Fail-Env**

\[
S; \{ \text{env}^* (M \cup r) \doteq \emptyset_{\text{env}} \} \cup P
\]

if \( M \) is nonempty
Solving Equations with Binding-Chains

**Dec-Chain**
\[
\{ \text{env}^*(\{s_1\} \cup M_1 \cup r_1) \triangleq \text{env}^*(\text{Ch}(x, y, l) \cup M_2 \cup r_2) \} \cup P
\]
non-deterministically select \( s_1 \) from \( M_1 \)
then guess position where \( s_1 \) can be equated with chain binding:
1) \( \{ l \triangleq 1, s_1 \triangleq \text{bind}(y, A(\text{var}(x))), \text{env}^*(M_1 \cup r_1) \triangleq \text{env}^*(M_2 \cup r_2) \} \cup P \)
2) \( \{ l \triangleq 1 + l_1, s_1 \triangleq \text{bind}(z, A(\text{var}(x))), \text{env}^*(M_1 \cup r_1) \triangleq \text{env}^*(\text{Ch}(z, y, l_1) \cup M_2 \cup r_2) \} \cup P \)
3) \( \{ l \triangleq l_1 + 1, s_1 \triangleq \text{bind}(y, A(\text{var}(z))), \text{env}^*(M_1 \cup r_1) \triangleq \text{env}^*(\text{Ch}(x, z, l_1) \cup M_2 \cup r_2) \} \cup P \)
4) \( \{ l \triangleq l_1 + 1 + l_2, s_1 \triangleq \text{bind}(z_2, A(\text{var}(z_1))), \text{env}^*(M_1 \cup r_1) \triangleq \text{env}^*(\text{Ch}(x, z_1, l_1) \cup \text{Ch}(z_2, y, l_2) \cup M_2 \cup r_2) \} \cup P \)

\( z, z_1, z_2 \) are fresh variables of sort \( \text{BV} \) and \( A \) is either a fresh context variable of class \( A \) if \( \text{Ch}=\text{NCh} \) or \([\cdot]\) if \( \text{Ch}=\text{VCh} \). \( l_1, l_2 \) are fresh \( \mathcal{N} \) variables.

C. Rau, M. Schmidt-Schauß: Unification in Lambda-Calculi with Variable Chains 15/1
Solving Equations with Binding-Chains

**LC Unification: Chain - Chain**

**U-Chain**

\[
\begin{align*}
\{ & \text{env}^* \left( \{ \text{bind}(x_1, s_1) \} \cup \text{VCh}(x_1, y_1, l_1) \cup M_1 \cup r_1 \right) \} \approx \\
& \text{env}^* \left( \{ \text{bind}(x_2, s_2) \} \cup \text{VCh}(x_2, y_2, l_2) \cup M_2 \cup r_2 \right) \} \cup P
\end{align*}
\]

1) \( l_1 \doteq l + l_1', l_2 \doteq l + l_2', \text{bind}(x_1, s_1) \doteq \text{bind}(x_2, s_2), \)
\[
\text{env}^* \left( \text{VCh}(z, y_1, l_1') \cup M_1 \cup r_1 \right) \doteq \text{env}^* \left( \text{VCh}(z, y_2, l_2') \cup M_2 \cup r_2 \right) \} \cup P;
\]
2) \( l_1 \doteq l_2, \text{bind}(x_1, s_1) \doteq \text{bind}(x_2, s_2), y_1 \doteq y_2, \)
\[
\text{env}^* \left( M_1 \cup r_1 \right) \doteq \text{env}^* \left( M_2 \cup r_2 \right) \} \cup P
\]

\( z \) and \( l, l_1', l_2' \) are fresh and \( \text{VCh}(z, y_1, l_1'), \text{VCh}(z, y_2, l_2') \) are disjoint

---

1. \( x_1 = s_1 \)
   \[
   \begin{array}{c}
   \text{VCh}(x_1,z,l) \\
   \text{VCh}(z,y_1,l_1')
   \end{array}
   \]

2. \( x_1 = s_1 \)
   \[
   \begin{array}{c}
   \text{VCh}(x_1,y_1,l_1) \\
   \text{VCh}(z,y_2,l_2')
   \end{array}
   \]
Results and Further Work

**Theorem**

For initial LR-forking problems the unification algorithm is

- **Sound** (solutions represent overlaps in LR)
- **Complete** (all overlaps in LR are computed)
- **Terminates** (halts for each equation set in IP)

and therefore computes a finite set of unifiers that represent all overlaps between transformations and normal order reductions in LR.

**Outlook**

- Close the overlaps
- Handle other steps in the correctness proof
- Extend implementation of the unification algorithm
- Application of the method to other calculi