GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Matching of Meta-Expressions with Recursive Bindings

David Sabel

Goethe-University Frankfurt am Main, Germany

UNIF 2017, Oxford, UK

† Research supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SA 2908/3-1.

Motivation

- **automated reasoning on programs and program transformations** w.r.t. operational semantics
- for program calculi with higher-order constructs and recursive bindings, e.g. **letrec-expressions**:

$$\texttt{letrec}\ x_1 = s_1; \ldots; x_n = s_n\ \texttt{in}\ t$$

- extended call-by-need lambda calculi with letrec that model core languages of **lazy functional programming languages** like Haskell

Application: Correctness of Program Transformations

Program transformation T is **correct** iff

$$\forall \ell \to r \in T\text{: } \forall \text{ contexts } C\text{: } C[\ell]{\downarrow} \iff C[r]{\downarrow}$$

where $\downarrow =$ successful evaluation w.r.t. standard reduction

Application: Correctness of Program Transformations

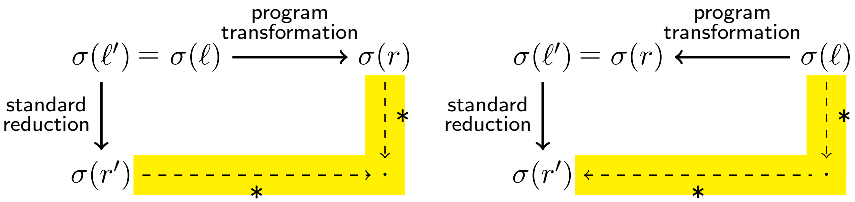
Program transformation T is **correct** iff

$$\forall \ell \to r \in T\text{: } \forall \text{ contexts } C\text{: } C[\ell]{\downarrow} \iff C[r]{\downarrow}$$

where $\downarrow =$ successful evaluation w.r.t. standard reduction

Diagram method to show correctness of transformations:

- Compute overlaps between standard reductions and program transformations (requires unification, see [SSS16, PPDP])
- Join the overlaps $\Rightarrow$ forking and commuting diagrams
- Induction using the diagrams (automatable, see [RSSS12, IJCAR])



GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\, Env\, \texttt{in}\, A \mid \texttt{letrec}\, \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\, A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \ \to\ T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

Requirements on the Meta-Language

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\ Env\ \texttt{in}\ A \mid \texttt{letrec}\ \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\ A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \to T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

Meta-syntax must be capable to represent:

- contexts of different classes
- environments Env_i,
- environment chains $\{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}$

Requirements on the Meta-Language

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\, Env\, \texttt{in}\, A \mid \texttt{letrec}\, \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\, A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \to T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

Meta-syntax must be capable to represent:

- contexts of different classes
- environments Env_i,
- environment chains $\{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}$

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\ \mathit{Env}\ \texttt{in}\ A \mid \texttt{letrec}\ \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, \mathit{Env}\ \texttt{in}\ A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ \mathit{Env}_1\ \texttt{in}\ \texttt{letrec}\ \mathit{Env}_2\ \texttt{in}\ e \to \texttt{letrec}\ \mathit{Env}_1, \mathit{Env}_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, \mathit{Env}\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, \mathit{Env}\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ \mathit{Env}\ \texttt{in}\ e] \to T[e]$ if $\mathit{LetVars}(\mathit{Env}) \cap \mathit{FV}(e) = \emptyset$

Meta-syntax must be capable to represent:

- contexts of different classes
- environments Env_i,
- environment chains $\{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}$

Requirements on the Meta-Language

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\, Env\, \texttt{in}\, A \mid \texttt{letrec}\ \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\, A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \ \to\ T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

Meta-syntax must be capable to represent:

- contexts of different classes
- environments Env_i,
- environment chains $\{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}$

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Variables	$x \in \textbf{Var} ::=$	X	(variable meta-variable)
	$\mid$	x	(concrete variable)
Expressions	$s \in \textbf{Expr} ::=$	S	(expression meta-variable)
	$\mid$	$\mathtt{var}\ x$	(variable)
	$\mid$	$(f\ r_1 \ldots r_{ar(f)})$	(function application)
		where r_i is $o_i, s_i,$ or x_i specified by f	
	$\mid$	$D[s]$	(context meta-variable)
	$\mid$	$\mathtt{letrec}\ env\ \mathtt{in}\ s$	(letrec-expression)
	$o \in \textbf{HExpr}^n ::=$	$x_1. \ldots x_n.s$	(higher-order expression)
Environments	$env \in \textbf{Env} ::=$	$\emptyset$	(empty environment)
	$\mid$	$E; env$	(environment meta-variable)
	$\mid$	$Ch[x, s]; env$	(chain meta-variable)
	$\mid$	$x{=}s; env$	(binding)

- Context variables D and Ch-variables have a context class $cl(D)$
- instances of $Ch[x, s]$: chains $x{=}\mathsf{D}_1[\mathtt{var}\ \mathsf{x}_1]; \mathsf{x}_1{=}\mathsf{D}_2[\mathtt{var}\ \mathsf{x}_2]; \ldots; \mathsf{x}_n{=}\mathsf{D}_n[s]$ where D_i are contexts of class $cl(Ch)$.

Binding and Scoping Constraints

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\, Env\, \texttt{in}\, A \mid \texttt{letrec}\, \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\, A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta)	$R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$
(SR,llet)	$\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$
(T,cpx)	$T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$
(T,gc)	$T[\texttt{letrec}\ Env\ \texttt{in}\ e] \to T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

restrictions on scoping and emptiness have to be respected, e.g.:

- (gc): Env must not be empty; side condition on variables
- (llet): $FV(Env_1) \cap LetVars(Env_2) = \emptyset$
- (cpx): x, y are not captured by C in $C[x]$

Binding and Scoping Constraints

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\, Env\, \texttt{in}\, A \mid \texttt{letrec}\, \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\, A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e]\ \to\ T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

restrictions on scoping and emptiness have to be respected, e.g.:

- (gc): Env must not be empty; side condition on variables
- (llet): $FV(Env_1) \cap LetVars(Env_2) = \emptyset$
- (cpx): x, y are not captured by C in $C[x]$

Binding and Scoping Constraints

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\ Env\ \texttt{in}\ A \mid \texttt{letrec}\ \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\ A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \to T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

restrictions on scoping and emptiness have to be respected, e.g.:

- (gc): Env must not be empty; side condition on variables
- (llet): $FV(Env_1) \cap LetVars(Env_2) = \emptyset$
- (cpx): x, y are not captured by C in $C[x]$

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Operational semantics of typical call-by-need calculi (excerpt)

Reduction contexts:

$A ::= [\cdot] \mid (A\ e)$

$R ::= A \mid \texttt{letrec}\ Env\ \texttt{in}\ A \mid \texttt{letrec}\ \{x_i{=}A_i[x_{i+1}]\}_{i=1}^{n-1}, x_n{=}A_n, Env\ \texttt{in}\ A[x_1]$

Standard-reduction rules and some program transformations:

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \rightarrow R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \rightarrow \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \rightarrow T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \rightarrow T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

restrictions on scoping and emptiness have to be respected, e.g.:

- (gc): Env must not be empty; side condition on variables
- (llet): $FV(Env_1) \cap LetVars(Env_2) = \emptyset$
- (cpx): x, y are not captured by C in $C[x]$

- A **constraint tuple** $\Delta = (\Delta_1, \Delta_2, \Delta_3)$ consists of
  Δ_1: set of context variables (non-empty context constraint)
  Δ_2: set of environment variables (non-empty environment constraint)
  Δ_3: set of pairs (s, d) (s an expression, d a context) (non-capture constraint)
- Ground substitution ρ **satisfies** $(\Delta_1, \Delta_2, \Delta_3)$ iff
  - $\rho(D) \neq [\cdot]$ for all $D \in \Delta_1$
  - $\rho(E) \neq \emptyset$ for all $E \in \Delta_2$
  - hole of $\rho(d)$ does not capture variables of $\rho(s)$, for all $(s, d) \in \Delta_3$

- A **constraint tuple** $\Delta = (\Delta_1, \Delta_2, \Delta_3)$ consists of
  Δ_1: set of context variables (non-empty context constraint)
  Δ_2: set of environment variables (non-empty environment constraint)
  Δ_3: set of pairs (s, d) (s an expression, d a context) (non-capture constraint)
- Ground substitution ρ **satisfies** $(\Delta_1, \Delta_2, \Delta_3)$ iff
  - $\rho(D) \neq [\cdot]$ for all $D \in \Delta_1$
  - $\rho(E) \neq \emptyset$ for all $E \in \Delta_2$
  - hole of $\rho(d)$ does not capture variables of $\rho(s)$, for all $(s, d) \in \Delta_3$
- A pair (s, Δ) is called a **constrained expression**
  $sem(s, \Delta) = \{\rho(s) \mid \rho(s)$ fulfills LVC and ρ satisfies $\Delta\}$
  (LVC = let variable convention, binders of the same environment are different)

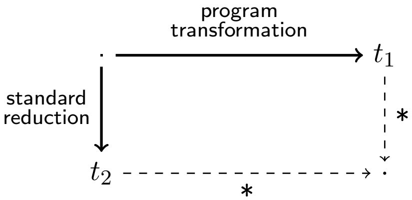
Example:

$s = \texttt{letrec}\ E_1\ \texttt{in}\ \texttt{letrec}\ E_2\ \texttt{in}\ S$

$\Delta = (\emptyset, \{E_1, E_2\}, \{(\texttt{letrec}\ E_2\ \texttt{in}\ S, \texttt{letrec}\ E_1\ \texttt{in}\ [\cdot])\})$

$sem(s, \Delta)$ = nested letrec-expressions with unused outer environment

Computing Diagrams



- t_1, t_2 are meta-expressions restricted by constraints ∇
- computing joins $\xrightarrow{*}$ requires abstract rewriting by rewrite rules $\ell \rightarrow_\Delta r$ with Δ restricting ℓ and r
- matching equations $\ell \trianglelefteq t$ together with constraint tuples ∇, Δ
- a matcher σ may instantiate ℓ but not t, i.e. $\sigma(\ell) = t$
- l contains instantiable meta-variables and t contains fixed meta-variables, denoted by $MV_I(\cdot)$ and $MV_F(\cdot)$

A **letrec matching problem** is a tuple $P{=}(\Gamma, \Delta, \nabla)$ where

- Γ is a set of matching equations $s \trianglelefteq t$ s.t. $MV_I(t) = \emptyset$
- $\Delta{=}(\Delta_1, \Delta_2, \Delta_3)$ is a constraint tuple (needed constraints);
- $\nabla{=}(\nabla_1,\nabla_2,\nabla_3)$ is a constraint tuple (given constraints), s.t. $MV_I(\nabla){=}\emptyset$ and ∇ is satisfiable.

Occurrence restrictions for instantiable meta variables:

- Each instantiable S-variable occurs at most twice in Γ
- Each E-, Ch-, D-variable occurs at most once in Γ

Matcher of $P = (\Gamma, \Delta, \nabla)$

A substitution σ is a **matcher of $P = (\Gamma, \Delta, \nabla)$** iff

- σ instantiates the instantiable variables and does not introduce new instantiable or fixed variables
- for any ground substitution ρ on $MV_F(P)$ that satisfies ∇ and where $\rho(\sigma(s))$ and $\rho(t)$ for $s \trianglelefteq t \in \Gamma$ fullfill the LVC:
  - $\rho(\sigma(s)) \sim_{let} \rho(t)$ for all $s \trianglelefteq t \in \Gamma$
  - the Δ-constraints hold
    ($\exists\ \rho_0$ with $\mathrm{Dom}(\rho_0) = MV_I(\rho(\sigma(\Delta)))$ s.t. $\rho_0(\rho(\sigma(\Delta)))$ is satisfied.)

$\sim_{let}$ = syntactic equality upto permuting bindings in environments

Theorem (NP-Hardness)

The decision problem whether a matcher for a letrec matching problem exists is **NP-hard**.

Proof by a reduction from MONOTONE ONE-IN-THREE-3-SAT.

Sketch: For each clause $C_i = \{S_{i,1}, S_{i,2}, S_{i,3}\}$, add the matching equation

$$\begin{array}{rl} & \texttt{letrec}\ Y_{i,1} = S_{i,1};\ \ Y_{i,2} = S_{i,2};\ \ Y_{i,3} = S_{i,3}\ \texttt{in}\ c \\ \trianglelefteq & \texttt{letrec}\ \mathsf{y}_{i,1} = \mathit{false};\ \mathsf{y}_{i,2} = \mathit{false};\ \mathsf{y}_{i,3} = \mathit{true}\ \texttt{in}\ c \end{array}$$

Matching Algorithm MatchLRS

Intermediate **data structure** of the algorithm: $(Sol, \Gamma, \Delta, \nabla)$ where

- Sol is a computed substitution
- Γ is a set of equations
- $(\Delta_1, \Delta_2, \Delta_3)$ are needed constraints
- $(\nabla_1, \nabla_2, \nabla_3)$ are given constraints

Input:
For $P = (\Gamma, \Delta, \nabla)$, MatchLRS starts with $(Id, \Gamma, \Delta, \nabla)$

Output (on each branch):
$Fail$ or final state $(Sol, \emptyset, \Delta, \nabla)$

Matching Algorithm MatchLRS: Rules

Inference rules of the form

$$\frac{\text{State}}{\text{State}_1 \mid \ldots \mid \text{State}_n}$$

Rule application is non-deterministic:

- don't care non-determinsm between the rules
- don't know non-determinism between $\text{State}_1 \mid \ldots \mid \text{State}_n$

Selection of Rules (1)

Solving an expression-variable:

$$\frac{(Sol,\Gamma\dot\cup\{S \trianglelefteq s\},\Delta)}{(Sol\circ\{S\mapsto s\},\Gamma[s/S],\Delta[s/S])}$$

Decomposition of letrec:

$$\frac{\Gamma\dot\cup\{\texttt{letrec}\ env\ \texttt{in}\, s \trianglelefteq \texttt{letrec}\ env'\ \texttt{in}\, t\}}{\Gamma\dot\cup\{env \trianglelefteq env', s \trianglelefteq t\}}$$

Prefix-rule for contexts: D' is a prefix of D

$$\frac{(Sol, \Gamma\dot\cup\{D[s] \trianglelefteq D'[s']\}, \Delta, \nabla)}{(Sol \circ \sigma, \Gamma\dot\cup\{D''[s] \trianglelefteq s'\}, \Delta\sigma, \nabla)} \text{ if } D \in \Delta_1 \iff D' \in \nabla_1 \text{ and } cl(D') \leq cl(D)$$

s.t. $\sigma=\{D \mapsto D'[D'']\}, cl(D'')=cl(D)$

Selection of Rules (2)

$$\frac{(Sol, \Gamma \dot\cup \{env \trianglelefteq b; env'\}, \Delta, \nabla)}{\begin{array}{l}\bigvee_{\forall b' : env = b'; env''} (Sol, \Gamma \dot\cup \{b' \trianglelefteq b, env'' \trianglelefteq env'\}, \Delta, \nabla) \\ \mid \bigvee_{\forall E : env = E; env''} (Sol \circ \sigma, \Gamma \dot\cup \{E'; env'' \trianglelefteq env'\}, \Delta\sigma, \nabla) \text{ where } \sigma = \{E \mapsto b; E'\} \\ \mid \bigvee_{\forall Ch : env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma \dot\cup \{y.D[s] \trianglelefteq b, env'' \trianglelefteq env'\}, \Delta\sigma, \nabla) \\ \quad \text{where } \sigma = \{Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].D[\cdot_2]\} \text{ and } cl(D) = cl(Ch) \\ \mid \bigvee_{\forall Ch : env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma \dot\cup \{y.D[X] \trianglelefteq b, Ch_2[X, s]; env'' \trianglelefteq env'\}, \Delta\sigma, \nabla) \\ \quad \text{where } \sigma = \{Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].D[X]; Ch_2[X, \cdot_2]\}, cl(D) = cl(Ch_2) = cl(Ch) \\ \mid \bigvee_{\forall Ch : env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma \dot\cup \{Y = D_1[X] \trianglelefteq b, Ch_1[y, D_2[Y]]; Ch_2[X, s]; env'' \trianglelefteq env'\}, \Delta\sigma, \nabla) \\ \quad \text{where } \sigma = \{Ch[\cdot_1, \cdot_2] \mapsto Ch_1[\cdot_1, D_2[Y]]; Y = D_1[X]; Ch_2[X, \cdot_2]\}, cl(D_i) = cl(Ch_i) = cl(Ch) \\ \mid \bigvee_{\forall Ch : env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma \dot\cup \{X_1 = D[s] \trianglelefteq b, Ch_1[y, D'[X_1]]; env'' \trianglelefteq env'\}, \Delta\sigma, \nabla) \text{ where} \\ \quad \sigma = \{Ch[\cdot_1, \cdot_2] \mapsto Ch_1[\cdot_1, D'[X_1]]; X_1.D[\cdot_2]\}, cl(D) = cl(D') = cl(Ch_1) = cl(Ch)\end{array}}$$

environment with at least one binding b on the rhs of the equation

Selection of Rules (2)

$$(Sol, \Gamma \dot\cup \{env \trianglelefteq b; env'\}, \Delta, \nabla)$$

$\Big|_{\forall b': env = b'; env''} (Sol, \Gamma\dot\cup\{b' \trianglelefteq b, env'' \trianglelefteq$

b equals a binding b' on the lhs

$| \Big|_{\forall E: env = E; env''} (Sol \circ \sigma, \Gamma\dot\cup\{E'; env'' \trianglelefteq$

b is part of an environment variable E on the lhs

$| \Big|_{\forall Ch: env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma\dot\cup\{y.D[s] \trianglelefteq b, env'' \trianglelefteq env'\}, \Delta\sigma, \nabla)$

where $\sigma = \{Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].D[\cdot_2]\}$ and $cl(D) = cl(Ch)$

$| \Big|_{\forall Ch: env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma\dot\cup\{y.D[X] \trianglelefteq$

where $\sigma = \{Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1]$

$| \Big|_{\forall Ch: env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma\dot\cup\{Y = D_1[X$ … $\nabla)$

where $\sigma = \{Ch[\cdot_1, \cdot_2] \mapsto Ch_1[$ … $Ch)$

$| \Big|_{\forall Ch: env = Ch[y,s]; env''} (Sol \circ \sigma, \Gamma\dot\cup\{X_1 = D[s]$

$\sigma = \{Ch[\cdot_1, \cdot_2] \mapsto Ch_1[\cdot_1, D'[$

b is part of a chain variable Ch on the lhs

4 cases:

- *chain consists of the single binding b*
- *b is a prefix of the chain*
- *b is an infix of the chain*
- *b is a suffix of the chain*

environment with at least one binding b on the rhs of the equation

Usual cases:

- Γ not empty, but no matching rule applicable
  Examples:
  - $f\ s_1\ \dots\ s_n \trianglelefteq g\ t_1\ \dots\ t_m$, or
  - $D[s] \trianglelefteq D'[t]$ and $cl(D) < cl(D')$.

Extraordinary cases:

- $(Sol, \emptyset, \Delta, \nabla)$ but for some s in an input equation $s \trianglelefteq t$, $Sol(s)$ violates the LVC
- NCC-implication check fails:
  - check that **given constraints ∇ imply needed constraints Δ**
  - also infers constraints from the LVC for input expressions

  Example: $\texttt{letrec}\ X_1 = S_1; X_2 = S_2\ \texttt{in}\ \dots$ implies validity of the non-capture constraint $(\texttt{var}\ X_1, \lambda X_2.[])$

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Theorem

$MatchLRS$ is **sound and complete**, i.e.

- (soundness) if $MatchLRS$ delivers $S = (Sol, \emptyset, \Delta, \nabla)$ for input P, then Sol is a matcher of P; and
- (completeness) if $P = (\Gamma, \Delta, \nabla)$ has a matcher σ, then there exists an output $(\sigma, \emptyset, \Delta_S, \nabla_S)$ of $MatchLRS$ for input P.

Theorem

$MatchLRS$ runs **in NP-time**.
The letrec matching problem is **NP-complete**.

Conclusion

- Sound and complete matching algorithm for LRSX
- Designed to represent program calculi with recursive bindings
- Letrec matching problem is NP-complete
- Automated computation of overlaps and joins for call-by-need core languages is possible
  Implementation: LRSX Tool (http://goethe.link/LRSXTOOL)

Conclusion

- Sound and complete matching algorithm for LRSX
- Designed to represent program calculi with recursive bindings
- Letrec matching problem is NP-complete
- Automated computation of overlaps and joins for call-by-need core languages is possible
  Implementation: LRSX Tool (http://goethe.link/LRSXTOOL)

Further work:

- join more cases by meta alpha-renaming (PPDP 2017, to appear)
- automated correctness of translations between program calculi