Matching of Meta-Expressions with Recursive Bindings

David Sabel*

Goethe-University, Frankfurt am Main, Germany
sabel@ki.informatik.uni-frankfurt.de

1 Motivation and Problem Description

We focus automated reasoning on program calculi with reduction semantics (see e.g. [8]), in particular, lambda-calculi with call-by-need evaluation and *letrec*-expressions (consisting of a set of recursive bindings and a body to reference the bindings) modelling the core of lazy functional programming languages like Haskell (see [1, 6]). The meta-language LRSX [5] is designed for this purpose. It uses higher-order function symbols (as they also occur in approaches using higher-order abstract syntax [3]), has a letrec-construct `letr`, meta-variables for environments, expressions, variables and contexts. Its syntax (see Fig. 1) is parametrized over context classes $\overline{K}$[1] and (higher-order) function symbols $\mathcal{F}$. A *ground* LRSX-expression (an LRS-*expression*) does not contain any meta-variable. *Contexts* are expressions with a hole $[\cdot] : \mathbf{HExpr}^0$.

The semantics of meta-variables is straight-forward except for chain-variables Ch where $Ch[x,s]$ of class $\mathcal{K}$ stands for $x.\mathsf{d}[s]$ or chains $x.\mathsf{d}_1[(\mathtt{var}\ \mathsf{x}_1)]; \mathsf{x}_1.\mathsf{d}_2[(\mathtt{var}\ \mathsf{x}_2)]; \ldots; \mathsf{x}_n.\mathsf{d}_n[s]$ with fresh variables x_i, and contexts d, d_i from the class $\mathcal{K}$. As a motivation for chain-variables, consider the reduction rule $\mathtt{letr}\ x_1{=}A_1[x_2], \ldots, x_{n-1}{=}A_n[x_n], x_n{=}(\lambda y.s_0)\ s_1\ \mathtt{in}\ A'[x_1] \rightarrow$ $\mathtt{letr}\ x_1{=}A_1[x_2], \ldots, x_{n-1}{=}A_n[x_n], x_n{=}(\mathtt{letr}\ y{=}s_1\ \mathtt{in}\ s_0)\ \mathtt{in}\ A'[x_1]$. It performs β-reduction with sharing at a needed position (where A,A_i are evaluation contexts) which is expressed by the informal notion $x_1{=}A_1[x_2], \ldots, x_{n-1}{=}A_n[x_n]$ for a chain of bindings of arbitrary length. In LRSX the left hand side of the rule can be represented as $\mathtt{letr}\,E; Ch[X_1, \mathsf{app}\,(\lambda Y.S_0)\,S_1]\,\mathtt{in}\,A[\mathtt{var}\,X_1]$ where Ch is a chain-variable of class $\mathcal{A}$. The example also shows that the meta-syntax requires a notion of contexts and context classes. The rule $\mathtt{letr}\ E_1\ \mathtt{in}\ \mathtt{letr}\ E_2\ \mathtt{in}\ s \rightarrow \mathtt{letr}\ E_1, E_2\ \mathtt{in}\ s$ joins two nested letrec-environments. The rule requires that scoping is respected, i.e. let-bindings of E_2 must not capture variables in E_1. That is why we use so-called *constrained expressions*:

Definition 1. *In a* constrained expression (s, Δ) *s is an* LRSX*-expression and* $\Delta = (\Delta_1, \Delta_2, \Delta_3)$ *is a* constraint tuple *s.t.* Δ_1 *is a set of context variables, called* non-empty context constraints*;* Δ_2 *is a set of environment variables, called* non-empty environment constraints*; and* Δ_3 *is a finite set of pairs* (t, d) *where* t *is an* LRSX*-expression and* d *is an* LRSX*-context, called* non-capture constraints *(NCCs). A ground substitution*[2] ρ satisfies Δ *iff* $\rho(D) \neq [\cdot]$ *for all* $D \in \Delta_1$*;* $\rho(E) \neq \emptyset$ *for all* $E \in \Delta_2$*; and* $Var(\rho(t)) \cap CV(\rho(d)) = \emptyset$ *for all* $(t,d) \in \Delta_3$*. The* concretizations *of* (s,Δ) *are* $\gamma(s,\Delta) := \{\rho(s) \mid \rho$ *is a ground substitution,* $\rho(s)$ *fulfills the LVC*[3]*,* ρ *satisfies* $\Delta\}$.

In this paper we treat matching of constrained expressions against constrained expressions. An application is to apply rewrite rules to constrained expressions which is necessary to

*This research is supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SA2908/3-1.

[1] A *context class* $\mathcal{K} \in \overline{K}$ is a set of contexts. To ease reading, we use $\overline{K} = \{Triv, \mathcal{A}, \mathcal{T}, \mathcal{C}\}$, where *Triv* consists of the empty context only, in $\mathcal{C}$-contexts the hole is allowed everywhere, $\mathcal{T}$ contexts have the hole not below a higher-order binder $x.s$, and the path to the hole in $\mathcal{A}$-contexts always uses strict positions of function symbols. We use the ordering $Triv < \mathcal{A} < \mathcal{T} < \mathcal{C}$, since $\mathcal{C} \supseteq \mathcal{T} \supseteq \mathcal{A} \supseteq Triv$.

[2] A substitution ρ is *ground* iff it maps all variables in $\mathtt{Dom}(\rho)$ to LRS-expressions.

[3] s satisfies the *let variable convention (LVC)* iff no binder occurs twice in the same `letr`-environment.

$$\begin{array}{ll} s \in \mathbf{HExpr}^0 ::= S \mid D[s] \mid \mathtt{letr}\ env\ \mathtt{in}\ s \mid f\ r_1 \ldots r_{ar(f)} & f \in \mathcal{F} ::= \mathtt{var} \mid \lambda \mid \ \ldots \\ \qquad \text{where } r_i \in \mathbf{HExpr}^k \text{ if } oar(f)(i) = k \geq 0, & x \in \mathbf{Var} ::= X \mid \mathtt{x} \\ \qquad \text{and } r_i \in \mathbf{Var}, \text{ if } oar(f)(i) = \mathbf{Var}. & b \in \mathbf{Bind} ::= x.s \text{ where } s \in \mathbf{HExpr}^0 \\ s \in \mathbf{HExpr}^n ::= x.s_1 \text{ if } s_1 \in \mathbf{HExpr}^{n-1}, n \geq 1 & env \in \mathbf{Env} ::= \emptyset \mid E; env \mid Ch[x,s]; env \mid b; env \end{array}$$

Figure 1: Syntax of LRSX: X, S, D, E, Ch are meta-variables. **Var** is a countably-infinite set of variables, $\mathbf{HExpr}^o$ are higher-order expressions of order $o \in \mathbb{N}_0$, **Env** are letr-environments, and **Bind** are letr-bindings. Every $f \in \mathcal{F}$ has a syntactic type $f : \tau_1 \to \ldots \to \tau_{ar(f)} \to \mathbf{HExpr}^0$, where τ_i may be **Var**, or $\mathbf{HExpr}^{k_i}$; the $ar(f)$-tuple $\langle \delta_1, \ldots, \delta_{ar(f)} \rangle$ is called the *order arity* $oar(f)$, where $\delta_i = k_i \in \mathbb{N}_0$, or $\delta_i = \mathbf{Var}$, depending on τ_i. For $f \in F$, $sp(f) \subseteq \{i \mid 1 \leq i \leq ar(f), oar(f)(i) = 0\}$ denotes the *set of strict positions of* f. We fix $\mathtt{var} : \mathbf{Var} \to \mathbf{HExpr}^0$ and $\lambda : \mathbf{HExpr}^1 \to \mathbf{HExpr}^0$. The *scope* of x is r in $x.r$ and the *scope* of x in $\mathtt{letr}\ x.s; env\ \mathtt{in}\ s'$ or $\mathtt{letr}\ Ch[x,s]; env\ \mathtt{in}\ s'$ is s, env, Ch and s'. A context variable D has a class $cl(D) \in \overline{K}$ and a Ch-variable has a class $cl(Ch) \in \{Triv, \mathcal{A}\}$. With $Var(r)$ we denote the set of variables in r and $LV(env)$ denotes the let-bound variables in environment env. For ground context $\mathtt{d}$, $CV(\mathtt{d})$ (the *captured variables*) denotes the variables which become bound if plugged into the hole of $\mathtt{d}$. The reflexive-transitive closure of permuting bindings in a letr-environment is denoted with $\sim_{let}$.

compute joins for critical pairs that occur in the diagram method (see [6] and also [7, 2]) – a syntactic method to prove the correctness of program transformations. Here critical pairs stem from overlapping left hand sides of calculus reduction steps with left or right hand sides of transformation rules. The overlaps are computed on constrained meta-expressions using the unification algorithm for LRSX-expressions from [5]. For computing joins, the transformations and reduction steps have to be applied to the constrained expressions where a requirement is that all steps must be applicable to each instance of the constrained expressions. This leads to the following matching problem with two kinds of meta-variables: Usually matching means to solve directed equations of the form $s \trianglelefteq t$ where s contains meta-variables and t is a ground expression. However, in our equations s is a meta-expression with *instantiable meta-variables* and t contains meta-variables which are treated like "meta-constants" (called *fixed meta-variables*). To distinguish the meta-variables, we use blue font for instantiable meta-variables and red font and underlining for fixed meta-variables. With $MV_I(\cdot)$ and $\underline{MV_F}(\cdot)$ we compute the sets.

Definition 2. *A* letrec matching problem *(LMP) 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.* $MV_I(\nabla_i)=\emptyset$ *for* $i = 1, 2, 3$*,* ∇ *is satisfiable, for all expressions in* Γ*, the LVC holds, and every instantiable variable of kind* S *(*E*,* Ch*,* D *, resp.) occurs at most twice (once, resp.) in* Γ*. A* matcher *of* P *is a substitution* σ *where* $\mathtt{Dom}(\sigma) = MV_I(\Gamma)$*,* $MV_I(\sigma(s)) = \emptyset$ *and* $\underline{MV_F}(\sigma(s)) \subseteq \underline{MV_F}(P)$ *for all* $s \trianglelefteq t \in \Gamma$*, s.t. for any ground substitution* ρ *with* $\mathtt{Dom}(\rho) = \underline{MV_F}(P)$ *which satisfies* ∇*,* $\rho(\sigma(s)), \rho(t)$ *fulfill the LVC for all* $s \trianglelefteq t \in \Gamma$*, we have* $\rho(\sigma(s)) \sim_{let} \rho(t)$ *for all* $s \trianglelefteq t \in \Gamma$ *and there exists a ground substitution* ρ_0 *with* $\mathtt{Dom}(\rho_0) = MV_I(\rho(\sigma(\Delta)))$ *s.t.* $\rho_0(\rho(\sigma(\Delta)))$ *is satisfied.*

Example 3. *The LMP* $(\{s \trianglelefteq t\},\Delta,\nabla)$ *with* $s = \mathtt{letr}\ E_1\ \mathtt{in}\ S_1$*,* $t = \mathtt{letr}\ \underline{E_2}\ \mathtt{in}\ \underline{S_2}$*,* $\Delta = (\emptyset, \{E_1\}, \{(S_1, \mathtt{letr}\ E_1\ \mathtt{in}\ [\cdot])\})$*, and* $\nabla = (\emptyset, \{\underline{E_2}\}, \emptyset)$ *has no matcher: The substitution* $\sigma = \{E_1 \mapsto \underline{E_2}, S_1 \mapsto \underline{S_2}\}$ *is not a matcher, since for instance, for* $\rho = \{\underline{E_2} \mapsto \mathtt{x.var\ x}, \underline{S_2} \mapsto \mathtt{var\ x}\}$ *the NCC* $\rho(\sigma((S_1, \mathtt{letr}\ E_1\ \mathtt{in}\ [\cdot]))) = (\mathtt{var\ x}, \mathtt{letr\ x.var\ x\ in}\ [\cdot])$ *is violated. However, the substitution* σ *is a matcher of the LMP* $(s \trianglelefteq t, \Delta, \nabla')$ *with* $\nabla' = (\emptyset, \{\underline{E_2}\}, \{(\underline{S_2}, \mathtt{letr}\ \underline{E_2}\ \mathtt{in}\ [\cdot])\})$.

The unification algorithm in [5] cannot be reused for matching, since its occurrence restrictions

$$\text{(SolX)}\ \frac{(Sol,\Gamma\dot\cup\{X \trianglelefteq x\},\Delta)}{(Sol\circ\{X\mapsto x\},\Gamma[x/X],\Delta[x/X])} \quad \text{(SolS)}\ \frac{(Sol,\Gamma\dot\cup\{S \trianglelefteq s\},\Delta)}{(Sol\circ\{S\mapsto s\},\Gamma[s/S],\Delta[s/S])} \quad \text{(DecH)}\ \frac{\Gamma\dot\cup\{x.s \trianglelefteq y.t\}}{\Gamma\dot\cup\{x \trianglelefteq y, s \trianglelefteq t\}}$$

$$\text{(DecL)}\ \frac{\Gamma\dot\cup\{\texttt{letr}\,env\,\texttt{in}\,s \trianglelefteq \texttt{letr}\,env'\,\texttt{in}\,t\}}{\Gamma\dot\cup\{env \trianglelefteq env', s \trianglelefteq t\}} \quad \text{(DecF)}\ \frac{\Gamma\dot\cup\{f\,s_1\ldots s_n \trianglelefteq f\,t_1\ldots t_n\}}{\Gamma\dot\cup\{s_1 \trianglelefteq t_1,\ldots,s_n \trianglelefteq t_n\}} \quad \text{(DecD)}\ \frac{\Gamma\dot\cup\{\underline{D}[s] \trianglelefteq \underline{D}[t]\}}{\Gamma\dot\cup\{s \trianglelefteq t\}}$$

$$\text{(CxPx)}\ \frac{(Sol,\Gamma\dot\cup\{D[s] \trianglelefteq \underline{D'}[s']\},\Delta,\nabla)}{(Sol\circ\sigma,\Gamma\dot\cup\{D''[s] \trianglelefteq s'\},\Delta\sigma,\nabla)\ \text{s.t. } \sigma=\{D\mapsto\underline{D'}[D'']\}, cl(D'')=cl(D)} \quad \text{if } D\in\Delta_1 \iff \underline{D'}\in\nabla_1 \text{ and } cl(\underline{D'}) \le cl(D)$$

$$\text{(ElX)}\ \frac{\Gamma\dot\cup\{x \trianglelefteq x\}}{\Gamma} \quad \text{(CxCG)}\ \frac{(Sol,\Gamma\dot\cup\{D[s] \trianglelefteq \underline{D'}[s']\},\Delta,\nabla)}{(Sol\circ\sigma,\Gamma\dot\cup\{s \trianglelefteq \underline{D'}[s']\},\Delta\sigma,\nabla)\ \text{where } \sigma=\{D\mapsto[\cdot]\}} \quad \text{if } D\notin\Delta_1,\ cl(\underline{D'}) > cl(D)$$

$$\text{(ElS)}\ \frac{\Gamma\dot\cup\{\underline{S} \trianglelefteq \underline{S}\}}{\Gamma} \quad \text{(CxGuess)}\ \frac{(Sol,\Gamma\dot\cup\{D[s] \trianglelefteq t\},\Delta,\nabla)}{(Sol\circ\{D\mapsto[\cdot]\},\Gamma\dot\cup\{s \trianglelefteq t\},\Delta[[\cdot]/D],\nabla) \mid (Sol,\Gamma\dot\cup\{D[s] \trianglelefteq t\},(\Delta_1\dot\cup\{D\},\Delta_2,\Delta_3),\nabla)} \quad \text{if } D\notin\Delta_1,\ t\neq\underline{D'}[s] \text{ with } \underline{D'}\notin\nabla_1 \text{ or } cl(\underline{D'}) > cl(D)$$

$$\text{(CxF)}\ \frac{(Sol,\Gamma\dot\cup\{D[s] \trianglelefteq f\ s_1\ldots s_n\},\Delta,\nabla)}{\Big|\ (Sol\circ\sigma_i,\Gamma\dot\cup\{D'[s] \trianglelefteq s_i\},\Delta\sigma_i,\nabla)\ \text{s.t. } D',X_1,\ldots,X_m \text{ are fresh, } cl(D')=cl(D), \text{ and } \sigma_i=\{D\mapsto f\,s_1\ldots s_{i-1}\,X_1\ldots X_m.D'\,s_{i+1}\ldots s_n\},} \quad \text{if } D\in\Delta_1$$

$i\in I$, where $I=sp(f)$, if $cl(D)=\mathcal{A}$, $I=\{i \mid oar(f)(i)=0\}$ if $cl(D)=\mathcal{T}$, $I=\{i \mid oar(f)(i)\neq V\}$ if $cl(D)=\mathcal{C}$

$$\text{(CxL)}\ \frac{(Sol,\Gamma\dot\cup\{D[s] \trianglelefteq \texttt{letr}\ env\ \texttt{in}\ s'\},\Delta,\nabla)}{(Sol\circ\sigma,\Gamma\cup\{D'[s] \trianglelefteq s'\},\Delta\sigma,\nabla)\ \text{s.t. } \sigma=\{D\mapsto\texttt{letr}\ env\ \texttt{in}\ D'\}, cl(D')=cl(D)} \quad \text{if } D\in\Delta_1,\ cl(D)\ge T$$

$$\mid (Sol\circ\sigma,\Gamma\cup\{E;Ch[X,D'[s]] \trianglelefteq env\},\Delta\sigma,\nabla)\ \text{s.t. } \sigma=\{D\mapsto\texttt{letr}\ E;Ch[X,D']\,\texttt{in}\,s'\}, cl(D')=cl(D), cl(Ch)=\mathcal{A}$$

Figure 2: Rules of MatchLRS for variable, expression, and binding equations

are too strong and it cannot infer whether the given constraints imply the needed constraints.

The additional substitution ρ_0 in the definition of a matcher is needed for the case that a transformation introduces "fresh" variables. E.g., the rewrite rule $\texttt{letr}\ X.c\ S_1\ \texttt{in}\ S_2 \to \texttt{letr}\ X.c\ (\texttt{var}\ Y);Y.S_1\ \texttt{in}\ S_2$ requires NCCs $\{(\texttt{var}\ X,\lambda Y.[\cdot]),(S_1,\lambda Y.[\cdot]),(S_2,\lambda Y.[\cdot])\}$ to ensure that Y is fresh. Matching the left hand side of the rule against $\texttt{letr u}.c\ (\texttt{var y})\ \texttt{in var u}$, will not instantiate the variable Y. After instantiation, the NCCs become $\{(\texttt{var u},\lambda Y.[\cdot]),(\texttt{var v},\lambda Y.[\cdot])\}$. Validity depends on the instantiation of Y. The definition of a matcher allows us to choose an instance that satisfies the constraints (e.g. $\rho_0=\{Y\mapsto \mathtt{w}\}$). Any instantiation which satisfies the NCCs is valid, and thus to use matching for symbolic reduction, we can also keep the constraints (instead of using a ground instance) and add them to the given constraints on the result.

2 Solving the Letrec Matching Problem

We present the algorithm MatchLRS to solve LMPs[4]. A *state* is a tuple $(Sol,\Gamma,\Delta,\nabla)$ where Sol is a computed substitution and (Γ,Δ,∇) is a LMP s.t. Γ consists of expression-, environment-, binding-, and variable-equations. For (Γ,Δ,∇), the state is initialized as $(Id,\Gamma,\Delta,\nabla)$. The output is either a final state $(Sol,\emptyset,\Delta,\nabla)$ or *Fail*. In Figs. 2 and 3 are the rules of MatchLRS where only necessary components of the states are shown. They are inference rules $\frac{S}{S_1 \mid \ldots \mid S_n}$ s.t. for state S, the algorithm (don't know) non-deterministically branches into $S_1,\ldots,S_n$.

[4]MatchLRS is implemented as a part of the LRSX Tool (goethe.link/LRSXTOOL) – a tool to automatically prove the correctness of program transformations.

Application of rules is don't care non-determinism. Rules (SolveX) and (SolveS) solve, and (ElX) and (ElS) eliminate an expression equation. Rules (DecF), (DecH), (DecL), and (DecD) decompose function symbols, higher-order binders, bindings, letrec-expressions, and contexts. Other rules on expressions treat equations of the form $D[s] \trianglelefteq t$, where (CxPx) covers the case that t is $\underline{D'}[t']$ and $\underline{D'}$ is a prefix of D where D must be at least as general as $\underline{D'}$. If $\underline{D'}$ is non-empty, but D may be empty, then rule (CxGuess) is applicable. If the class of $\underline{D'}$ is strictly more general than the class of D, D must be instantiated by the empty context (rule (CxCG)). Rules (CxF) and (CxL) match the context variable against a function symbol or a `letr`-expression. Rules (EnvEm) and (ElE) eliminate, and (SolveE) solves an environment equation. Rule (EnvAE) solves a set of environment variables by instantiating them with $\emptyset$, where env is non-empty if $env = b; env'$, $env = \underline{Ch}[y, s]; env$, or $env = \underline{E'}; env$ with $\underline{E'} \in \nabla_2$. Rule (EnvB) is applicable if the right hand side of the equation contains a binding which may be matched against a binding, a part of a non-empty environment variable, or a part of a chain-variable, where four cases are possible: the binding coincides with, the binding is a prefix, a proper infix, or a suffix of the chain. Rule (EnvE) applies if the right hand side of an equation contains a fixed environment variable which has to be matched with a part of an instantiable variable. Rule (EnvC) covers the cases that a fixed chain-variable on the right hand side must be matched against the same variable on the left hand side, an instantiable environment variable, or and instantiable chain-variable.

For input $(\Gamma_I, \Delta_I, \nabla_I)$ and state $(Sol, \Gamma, \Delta, \nabla)$, MatchLRS delivers *Fail* if $\Gamma \neq \emptyset$ and no rule is applicable, or if $\Gamma = \emptyset$ and i) for $s \trianglelefteq t \in \Gamma_I$, $Sol(s)$ violates the LVC, or ii) the NCC-implication check (Def. 4) is invalid. For this check, we split NCCs into *atomic NCCs* (u, v) s.t. u, v are variables or meta-variables as $split_{ncc}(\mathcal{S}) := \bigcup_{(s,d)\in\mathcal{S}} Var_M(s) \times CV_M(d)$ where $Var_M(s) = MV_I(s) \cup \underline{MV_F}(s) \cup Var(s)$, and CV_M collects all concrete variables that capture variables of the context hole, and all meta-variables which may have concretizations that introduce capture variables.

Definition 4. *Let $(Sol, \emptyset, \Delta, \nabla)$ be a final state of* MatchLRS *for input $(\Gamma_I, \Delta_I, \nabla_I)$. The* NCC-implication check *is* valid *iff for all $(u, v) \in split_{ncc}(\Delta_3)$ one of the following cases holds*[5]*:*

1. $(u, v) \in split_{ncc}(\nabla_3) \cup NCC_{lvc}$ *or* $(u, v) = (\mathsf{x}, \mathsf{y})$ *where* $\mathsf{x} \neq \mathsf{y}$.
2. $u = v$ *and* $u = Ch$ *or* $u = D$ *or* $u = E$ *with* $E \notin \Delta_2$.
3. $u \neq v$ *and* $u \in \{Ch, S, D, E, X\}$ *or* $v \in \{Ch, D, E, X\}$.
4. $u = \underline{E}$ *or* $u = \underline{Ch}$ *with* $cl(\underline{Ch}) = Triv$ *and* $(u, u) \in split_{ncc}(\nabla_3) \cup NCC_{lvc}$.
5. $v \in \{\underline{E}, \underline{Ch}, \underline{D}\}$ *and* $(v, v) \in split_{ncc}(\nabla_3) \cup NCC_{lvc}$.
6. $(v, u) \in split_{ncc}(\nabla_3) \cup NCC_{lvc}$ *and* $(u, v) \in \{(\underline{X}, \mathsf{y}), (\mathsf{x}, \underline{Y}), (\underline{X}, \underline{Y}), (\mathsf{x}, \underline{D}), (\underline{X}, \underline{D}), (\mathsf{x}, \underline{E}), (\underline{X}, \underline{E}), (\mathsf{x}, \underline{Ch}), (\underline{X}, \underline{Ch}), (\underline{Ch_1}, \mathsf{x}), (\underline{Ch_1}, \underline{X}), (\underline{Ch_1}, \underline{E}), (\underline{Ch_1}, \underline{D}), (\underline{Ch_1}, \underline{Ch_2})\}$ *where* $cl(\underline{Ch_1}) = Triv$.

Example 5. *We apply* MatchLRS *for the LMP $(\{s \trianglelefteq t\}, \Delta, \nabla)$ with $s = \mathtt{letr}\ Ch[X, S_1]\ \mathtt{in}\ S_2$, $\Delta = (\Delta_1, \Delta_2, \Delta_3) = (\emptyset, \emptyset, \{(S_1, \lambda X.[\cdot])\})$, $t = \mathtt{letr}\ \underline{Y}.\mathsf{app}\ \underline{S_3}\ \underline{S_4}\ \mathtt{in}\ \underline{S_5}$, and $\nabla = (\nabla_1, \nabla_2, \nabla_3) = (\emptyset, \emptyset, \{(\underline{S_3}, \lambda\underline{Y}.[\cdot])\})$ where $cl(Ch) = \mathcal{A}$. Applying* (DecL) *and* (SolS)*, yields $(\{S_2 \mapsto \underline{S_5}\}, Ch[X, S_1] \trianglelefteq \underline{Y}.\mathsf{app}\ \underline{S_3}\ \underline{S_4}, \Delta, \nabla)$ and* (EnvB) *branches into four states, where all but the first case result in Fail, since they imply that Ch contains more than one binding. The remaining state is $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2]\}, X.A[S_1] \trianglelefteq \underline{Y}.\mathsf{app}\ \underline{S_3}\ \underline{S_4}, \Delta, \nabla)$. Applying* (DecH) *and* (SolX) *results in $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}\}, A[S_1] \trianglelefteq \mathsf{app}\ \underline{S_3}\ \underline{S_4}, \Delta, \nabla)$. Now* (CxGuess) *is applied and branches into two cases.*

[5] $NCC_{lvc} := \bigcup\{NCC_{lvc}(r) \mid r \in \{Sol(s), t\}, s \trianglelefteq t \in \Gamma_I\}$ represents atomic NCCs implied by the LVC where $NCC_{lvc}(t) = \{(x, y) \mid x.s; y.s'; env{\in}\mathcal{E} \lor x.s; \underline{Ch}[y, s']; env{\in}\mathcal{E} \lor \underline{Ch}[x, s]; y.s'; env{\in}\mathcal{E} \lor \underline{Ch}[x, s]; \underline{Ch'}[y, s']; env{\in}\mathcal{E}\} \cup \{(x, \underline{E}) \mid x.s; \underline{E}; env \in \mathcal{E} \lor \underline{Ch}[x, s]; \underline{E}; env \in \mathcal{E}\} \cup \{(x, \underline{Ch}) \mid x.s; \underline{Ch}[y, s]; env \in \mathcal{E} \lor \underline{Ch'}[x, s]; \underline{Ch}[y, s]; env{\in}\mathcal{E}\} \cup \{(\underline{Ch}, \underline{E}) \mid \underline{Ch}[y, s]; \underline{E}; env \in \mathcal{E},\ cl(\underline{Ch}) = Triv\} \cup \{(\underline{Ch_1}, \underline{Ch_2}) \mid \underline{Ch_1}[y, s]; \underline{Ch_2}[y', s']; env \in \mathcal{E},\ cl(\underline{Ch_1}) = Triv\}$ and $\mathcal{E}$ is the set of all `letr`-environments in t.

$$\text{(EnvAE)}\ \frac{(Sol,\Gamma\dot\cup\{E_1,\ldots,E_n \trianglelefteq \emptyset\},\Delta)}{(Sol\circ\sigma,\Gamma,\Delta\sigma)\ \text{s.t.}\ \sigma=\{E_i\mapsto\emptyset\}_{i=1}^{n}}\ \text{if}\ \forall i{:}E_i\not\in\Delta_2 \quad \text{(ElE)}\ \frac{\Gamma\dot\cup\{\underline{E};env_1 \trianglelefteq \underline{E};env_2\}}{\Gamma\dot\cup\{env_1 \trianglelefteq env_2\}} \quad \text{(EnvEm)}\ \frac{\Gamma\dot\cup\{\emptyset \trianglelefteq \emptyset\}}{\Gamma}$$

$$\text{(EnvE)}\ \frac{(Sol,\Gamma\dot\cup\{env \trianglelefteq \underline{E};env'\},\Delta,\nabla)}{\mathop{\Big|}\limits_{\forall E':env=E';env_1\ \text{and}\ \underline{E}\not\in\nabla_2 \implies E'\not\in\Delta_2}(Sol\circ\sigma,\Gamma\dot\cup\{E'';env_1 \trianglelefteq env'\},\Delta\sigma,\nabla)\ \text{with}\ \sigma=\{E'\mapsto E'';\underline{E}\}}\ \begin{array}{l}\text{if}\ env \neq \underline{E};env_1,\ \exists E{:}env=E;env_1\\ \text{s.t.}\ \underline{E}\not\in\nabla_2 \implies E\not\in\Delta_2\end{array}$$

$$\text{(SolveE)}\ \frac{(Sol,\Gamma\dot\cup\{E \trianglelefteq env\},\Delta,\nabla)}{(Sol\circ\sigma,\Gamma,\Delta\sigma,\nabla)\ \text{where}\ \sigma=\{E\mapsto env\}}\ \begin{array}{l}\text{if}\ E\in\Delta_2 \iff\\ env\ \text{is non-empty}\end{array}$$

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

$$\text{(EnvC)}\ \frac{(Sol,\Gamma\dot\cup\{env_1 \trianglelefteq \underline{Ch}[y,s];env_2\},\Delta,\nabla)}{\begin{array}{l}
\mathop{\Big|}\limits_{\forall \underline{Ch}:env_1=\underline{Ch}[y',s'];env_1'}(Sol\circ\sigma,\Gamma\dot\cup\{y' \trianglelefteq y,\ s' \trianglelefteq s,\ env_1' \trianglelefteq env_2\},\Delta\sigma,\nabla)\\
\mid \mathop{\Big|}\limits_{\forall E:env_1=E;env_1'}(Sol\circ\sigma,\Gamma\dot\cup\{E';env_1' \trianglelefteq env_2\},\Delta\sigma,\nabla)\ \text{where}\ \sigma=\{E\mapsto E';\underline{Ch}[y,s]\}\\
\mid \mathop{\Big|}\limits_{\forall Ch_1:env_1=Ch_1[y_1,s_1];env_1'\ \text{and}\ cl(Ch_1)\geq cl(\underline{Ch})}\ \mathop{\Big|}\limits_{\forall(d,t)\in split_{cl(Ch_1)}(s)}(Sol\circ\sigma,\Gamma\dot\cup\{env_1' \trianglelefteq env_2,y_1 \trianglelefteq y,s_1 \trianglelefteq t\},\Delta\sigma,\nabla)\ \text{with}\ \sigma=\{Ch_1[\cdot,\cdot]\mapsto\underline{Ch}[\cdot,d[\cdot_2]]\}\\
\mid \mathop{\Big|}\limits_{\forall Ch_1:env_1=Ch_1[y_1,s_1];env_1'\text{and}\ cl(Ch_1)\geq cl(\underline{Ch})}\ \mathop{\Big|}\limits_{\forall(d,t)\in split_{cl(Ch_1)}(s)}\begin{array}{l}(Sol\circ\sigma,\Gamma\dot\cup\{Ch_2[y_1,D[\mathtt{var}\ y]];env_1' \trianglelefteq env_2,s_1 \trianglelefteq t\},\Delta\sigma,\nabla)\\ \quad\text{where}\ \sigma=\{Ch_1[\cdot_1,\cdot_2]\mapsto Ch_2[\cdot_1,D[\mathtt{var}\ y]];\underline{Ch}[y,d[\cdot_2]]\},\ cl(D)=cl(Ch_2)=cl(Ch_1)\end{array}\\
\mid \mathop{\Big|}\limits_{\forall Ch_1\,:\,env_1=Ch_1[y_1,s_1];env_1'\text{and}\ cl(Ch_1)\geq cl(\underline{Ch})}\begin{array}{l}(Sol\circ\sigma,\Gamma\dot\cup\{Ch_2[X,s_1];env_1' \trianglelefteq env_2,D[\mathtt{var}\ X] \trianglelefteq s,y_1 \trianglelefteq y\},\Delta\sigma,\nabla)\\ \quad\text{where}\ \sigma=\{Ch_1[\cdot_1,\cdot_2]\mapsto\underline{Ch}[\cdot_1,s];Ch_2[X,\cdot_2]\}\ \text{and}\ cl(D)=cl(Ch_2)=cl(Ch_1)\end{array}\\
\mid \mathop{\Big|}\limits_{\forall Ch_1\,:\,env_1=Ch_1[y_1,s_1];env_1'\text{and}\ cl(Ch_1)\geq cl(\underline{Ch})}\begin{array}{l}(Sol\circ\sigma,\Gamma\dot\cup\{Ch_2[y_1,D[X]];Ch_3[Y,s_1];env_1' \trianglelefteq env_2,D_1[Y] \trianglelefteq s\},\Delta\sigma,\nabla)\\ \quad\text{where}\ \sigma=\{Ch_1[\cdot_1,\cdot_2]\mapsto Ch_2[\cdot_1,D[X]];\underline{Ch}[y,s];Ch_3[Y,\cdot_2]\}\ \text{and}\ cl(D)=cl(D_1)=cl(Ch_2)=cl(Ch_3)=cl(Ch_1)\end{array}
\end{array}}$$

Figure 3: Rules of MatchLRS for environment equations. In rule (EnvC) the function $split_{\mathcal{K}}$ is defined as follows: $split_{Triv}(t)=\{([\cdot],t)\}$; $split_{\mathcal{A}}(f\,s_1\ldots s_n)=\{([\cdot],(f\,s_1\ldots s_n))\}\cup\{(f\,s_1\ldots s_{i-1}\,d\,s_{i+1}\ldots s_n,s')\mid(d,s')\in split_{\mathcal{A}}(s_i),i\in sp(f)\}$; $split_{\mathcal{A}}(\underline{A}[s])=\{([\cdot],\underline{A}[s])\}\cup\{(\underline{A}[d],s')\mid(d,s')\in split_{\mathcal{A}}(s)\}$; and $split_{\mathcal{A}}(t)=\{([\cdot],t)\}$, if $t\neq(f\,s_1\ldots s_n)$ and $t\neq\underline{A}[s]$.

If A *is guessed as empty, then the next state is* $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}\}, S_1 \trianglelefteq \mathsf{app}\ \underline{S_3}\ \underline{S_4}, \Delta[\underline{Y}/X], \nabla)$*. Applying* (SolS) *yields* $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].[\cdot_2], X \mapsto \underline{Y}, S_1 \mapsto \mathsf{app}\ \underline{S_3}\ \underline{S_4}\}, \emptyset, \Delta', \nabla)$ *where* $\Delta' = (\emptyset, \emptyset, \{(\mathsf{app}\ \underline{S_3}\ \underline{S_4}, \lambda\underline{Y}.[\cdot])\})$*. Now the NCC-implication check fails since* $split_{ncc}(\Delta_3) = \{(\underline{S_3}, \underline{Y}), (\underline{S_4}, \underline{Y})\}$*,* $split_{ncc}(\nabla_3) = \{(\underline{S_3}, \underline{Y})\}$*, and* $NCC_{lvc} = \emptyset$ *and thus for* $(\underline{S_4}, \underline{Y}) \in split_{ncc}(\Delta_3)$ *none of the cases of Def. 4 holds.*

If A *is added to* Δ_1*, i.e. with* $\Delta'' = (\{A\}, \emptyset, \{(S_1, \lambda\underline{Y}.[\cdot])\})$ *we get* $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}\}, A[S_1] \trianglelefteq \mathsf{app}\ \underline{S_3}\ \underline{S_4}, \Delta'', \nabla)$*. Assuming that* $sp(\mathsf{app}) = \{1\}$*, rule* (CxF) *results in* $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}, A \mapsto \mathsf{app}\ A'\ \underline{S_4}\}, A'[S_1] \trianglelefteq \underline{S_3}, \Delta'', \nabla)$*. Applying* (CxGuess) *yields two cases: if* A' *is guessed to be non-empty, rule* (CxFailI) *is leads to Fail, and if* A' *is guessed to be empty, the next state is* $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}, A \mapsto \mathsf{app}\ [\cdot]\ \underline{S_4}\}, S_1 \trianglelefteq \underline{S_3}, \Delta'', \nabla)$ *and* (SolS) *results in* $(\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}, A \mapsto \mathsf{app}\ [\cdot]\ \underline{S_4}, S_1 \trianglelefteq \underline{S_3}\}, \emptyset, \Delta''', \nabla)$ *where* $\Delta''' = (\emptyset, \emptyset, \{(\underline{S_3}, \lambda\underline{Y}.[\cdot])\})$*. The NCC-implication check is valid since* $split_{ncc}(\Delta_3) = \{(\underline{S_3}, \underline{Y})\}$ *and* $(\underline{S_3}, \underline{Y}) \in split_{ncc}(\nabla_3)$*. Thus the algorithm delivers the matcher* $\{S_2 \mapsto \underline{S_5}, Ch[\cdot_1, \cdot_2] \mapsto [\cdot_1].A[\cdot_2], X \mapsto \underline{Y}, A \mapsto \mathsf{app}\ [\cdot]\ \underline{S_4}, S_1 \trianglelefteq \underline{S_3}\}$*.*

In the technical report [4] the following properties for MatchLRS and the LMP are proved:

Theorem 6. MatchLRS *is sound and complete, i.e. i) (soundness) if* MatchLRS *delivers* $S = (Sol, \emptyset, \Delta, \nabla)$ *for input* P*, then* Sol *is a matcher of* P*; and ii) (completeness) if a LMP* $P = (\Gamma, \Delta, \nabla)$ *has a matcher* σ*, then there exists an output* $(\sigma, \emptyset, \Delta_S, \nabla_S)$ *of* MatchLRS *for input* P*.* MatchLRS *runs in NP-time, and the letrec matching problem is NP-complete.*

3 Conclusion

Motivated by symbolically rewriting meta-expressions of the language `LRSX`, we formulated the letrec matching problem and developed the algorithm MatchLRS. We obtained soundness and completeness for MatchLRS and NP-completeness of the letrec matching problem. The presented algorithms are implemented in the LRSX Tool, and are part of an automated method to prove correctness of program transformations for program calculi expressible in `LRSX`.

Acknowledgments We thank the reviewers of UNIF 2017 for their valuable comments.

References

[1] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *POPL 1995*, pp. 233–246. ACM, 1995.

[2] E. Machkasova and F. A. Turbak. A calculus for link-time compilation. In *ESOP 2000*, LNCS 1782, pp. 260–274. Springer, 2000.

[3] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *PLDI 1988*, pp. 199–208. ACM, 1988.

[4] D. Sabel. Rewriting of higher-order-meta-expressions with recursive bindings. Frankfurter Informatik-Berichte 2017-1, Goethe-University Frankfurt am Main, 2017. http://goethe.link/fib-2017-1.

[5] M. Schmidt-Schauß and D. Sabel. Unification of program expressions with recursive bindings. In *PPDP 2016*, pp. 160–173. ACM, 2016.

[6] M. Schmidt-Schauß, M. Schütz, and D. Sabel. Safety of Nöcker's strictness analysis. *J. Funct. Programming*, 18(04):503–551, 2008.

[7] J. B. Wells, D. Plump, and F. Kamareddine. Diagrams for meaning preservation. In *RTA 2003*, LNCS 2706, pp. 88 –106. Springer, 2003.

[8] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, 1994.