Rewriting and Call-time choice: the HO case*

(Extended version)

Tech. Rep. UCM-SIC-3-08, 2008

F.J. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
fraguas@sip.ucm.es, jrodrigu@fdi.ucm.es, jaime@sip.ucm.es

Abstract. It is known that the behavior of non-deterministic functions with call-time choice semantics, present in current functional logic languages, is not well described by usual approaches to reduction like ordinary term rewriting systems or λ-calculus. The presence of HO features makes things more difficult, since reasoning principles that are essential in a standard (i.e., deterministic) functional setting, like extensionality, become wrong. In this paper we propose *HOlet*-rewriting, a notion of rewriting with local bindings that turns out to be adequate for programs with HO non-deterministic functions, as it is shown by strong equivalence results with respect to *HOCRWL*, a previously existing semantic framework for such programs. In addition, we give a sound and complete notion of *HOlet*-narrowing, we show by a case study the usefulness of the achieved combination of semantic and reduction notions, and finally we prove within our framework that a standard approach to the implementation of HO features, namely translation to FO, is still valid for HO nondeterministic functions.

1 Introduction

Functional logic programming (FLP, for short; see [12, 14] for surveys) integrates features of logic programming and functional programming. Typically FLP adopts mostly a (lazy) functional style, thus making intensive use of higher order (HO) functions. However, most of the work about FLP focuses on first order (FO) aspects of programs, thus limiting the applicability of results.

This is not a satisfactory situation, especially taking into account that the presence of functions that are at the same time HO and non-deterministic leads to somehow surprising behaviors, as shown by the example we sent recently to the Curry mailing list [13]:

Example 1. Consider the following program computing with natural numbers represented by the constructors 0 and $s/1$, and where + is defined as usual.

* This work has been partially supported by the Spanish projects Merit-Forms-UCM (TIN2005-09207-C03-03) and Promesas-CAM (S-0505/TIC/0407).

```
g X -> 0          f -> g          f' X -> f X
h X -> s 0        f -> h

fadd F G X -> (F X) + (G X)       fdouble F -> fadd F F
```

Notice that f and f' are non-deterministic functions that are (by definition of f') extensionally equivalent; from the point of view of standard functional programming they should be seen as 'the same function'. However, consider the expressions *(fdouble f 0)* and *(fdouble f' 0)*. In modern FLP languages like Curry [16] or Toy [20], the possible values for *(fdouble f 0)* are *0, s (s 0)*, while *(fdouble f' 0)* can be in addition reduced to *s 0*.

This behavior corresponds to *call-time choice* [17, 11], the semantics for non-determinism adopted by those systems. Operationally call-time choice is very close to the *sharing* mechanism used in functional languages to implement lazy evaluation.

The example was sent[1] to point out that η-expansion and η-reduction are not valid for such systems, because extensionally equivalent functions (e.g., f and f') can be semantically distinguishable when put in the same context (e.g., *double* [ ] 0), a fact that does not happen neither in standard (i.e, deterministic) functional programs[2], nor in FO FLP. We remark also that with *run-time choice* [17, 11], f and f' will be indistinguishable (*double f 0* and *double f' 0* would both produce *0, s 0, s (s 0)* as possible results). Therefore, it is the combination *HO + Non-determinism + call-time choice* which makes things different.

That combination was addressed in *HOCRWL* [7, 8], an extension to HO of *CRWL* [3] [11], a semantic framework specifically devised for FLP with call-time choice semantics for non-determinism (see [27] for a survey of *CRWL* and its extensions). *HOCRWL* provides logic and model-theoretic semantics, based on an *intensional* view of functions, where different descriptions –in the form of *HO-patterns*– of the same extensional function are distinguished as different data. This allows expressive programs and is simpler than λ-calculus-based HO unification, which is an alternative approach followed in the logic programming setting [22]. Previous work on the intensional view of HO-FLP [10] did not consider non-determinism. Other works covering HO in FLP, [23, 15], consider orthogonal or inductively sequential (henceforth deterministic) systems; if extended directly to the non-deterministic case, they would realize run-time choice, as happens also with [4], where a type-based translation to FO in the spirit of [28, 9] is proposed. We remark also that [15] is close to the theory of HO rewriting [26], and therefore has η-expansion as a valid procedure, against the expected properties of the languages considered by ours. Finally, [1] copes with call-time choice but their approach to HO is again based on a FO-translation, in contrast to ours.

[1] As far as we know, it was the first time that this behavior was noticed.

[2] Although the addition of primitive functions not definable in the language like *seq* in Haskell [24] can also destroy extensionality.

[3] CRWL stands for *Constructor Based Rewriting Logic*.

A weak point of the original *(HO)CRWL*-way to FLP is that it does not come with a clear, simple notion of one-step reduction similar to one-step rewriting. In [19] we proposed *let-rewriting*, a notion of rewriting with local bindings adequate to FO *CRWL* semantics, and at the same time simpler and more abstract than other reduction notions based on term graph rewriting [25, 6] or natural operational semantics [1]. *Let*-rewriting was generalized to *let*-narrowing in [18].

Our aim in this work is to extend the notion of *let*-rewriting/narrowing to the HO case. We address various foundational aspects –definition of *HOlet*-rewriting and equivalence wrt the declarative semantics given by *HOCRWL* (Sect. 3), *HOlet*-narrowing and its soundness and completeness wrt *HOlet*-rewriting (Sect. 4)– and also more applied aspects, as are the use of our framework to language development (Sect. 5) or the proof of correctness within our framework of a scheme of translation to FO, the basis of a standard approach [28, 9, 4] to the implementation of HO stuff in FO settings.

There are still some other important issues –evaluation strategies (including concurrency), types, constraints– that have been left out of the scope of the paper. Finally, we are not inventing HO FLP, but only contributing to some aspects of its foundation. Therefore it is not our aim in this paper convincing of the practical interest of HO FLP: other documents [16, 27, 7, 4] contain enough evidences of that. Proofs can be found in an appendix.

2 Preliminaries: *HOCRWL*

We present here some basic notions and new results about *HOCRWL* [7].

2.1 Expressions, patterns and programs

We consider *function* symbols $f, g, \ldots \in FS$, *constructor* symbols $c, d, \ldots \in CS$, and *variables* $X, Y, \ldots \in \mathcal{V}$; each $h \in FS \cup CS$ has an associated *arity*, $ar(h) \in \mathbb{N}$; FS^n (resp. CS^n) is the set of function (resp. constructor) symbols with arity n. The notation $\overline{o}$ stands for tuples of any kind of syntactic objects o. The set of *applicative expressions* is defined by $Exp \ni e ::= X \mid h \mid (e_1\ e_2)$. As usual, application is left associative and outer parentheses can be omitted, so that $e_1\ e_2\ \ldots e_n$ stands for $((\ldots(e_1\ e_2)\ldots)\ e_n)$. The set of variables occurring in e is written by $var(e)$. A distinguished set of expressions is that of *patterns* $t, s \in Pat$, defined by: $t ::= X \mid c\ t_1 \ldots t_n \mid f\ t_1 \ldots t_m$, where $0 \leq n \leq ar(c), 0 \leq m < ar(f)$. Patterns are irreducible expressions playing the role of *values*. *FO-patterns*, defined by $FOPat \ni t ::= X \mid c\ t_1 \ldots t_n\ (n = ar(c))$, correspond to FO constructor terms, representing ordinary non-functional data-values. Partial applications of symbols $h \in FS \cup CS$ to other patterns are HO-patterns and can be seen as truly data-values representing functions from an *intensional* point of view. Examples of patterns with the signature of Ex. 1 are: *0, s X, s, f', fadd f' f'*. The last three are HO-patterns. Notice that *f, fadd f f* are not patterns since f is not a pattern ($ar(f) = 0$).

Expressions $X\ e_1 \ldots e_m$ ($m \geq 0$) are called *flexible* (*variable application* when $m > 0$). *Rigid* expressions have the form $h\ e_1 \ldots e_m$; moreover, they are *junk* if $h \in CS^n$ and $m > n$, *active* if $h \in FS^n$ and $m \geq n$, and *passive* otherwise.

Contexts are expressions with a hole defined as $Cntxt \ni \mathcal{C} ::= [\,] \mid \mathcal{C}\ e \mid e\ \mathcal{C}$. Application of $\mathcal{C}$ to e (written $\mathcal{C}[e]$) is defined by $[\,][e] = e$; $(\mathcal{C}\ e')[e] = \mathcal{C}[e]\ e'$; $(e'\ \mathcal{C})[e] = e'\ \mathcal{C}[e]$. Substitutions $\theta \in Subst$ are finite mappings from variables to expressions; $[X_i/e_i, \ldots, X_n/e_n]$ is the substitution which assigns $e_i \in Exp$ to the corresponding $X_i \in \mathcal{V}$. We will mostly use *pattern-substitutions* $PSubst = \{\theta \in Subst \mid \theta(X) \in Pat, \forall X \in \mathcal{V}\}$. We write ϵ for the identity substitution, $dom(\theta)$ for the domain of θ, and $vRan(\theta) = \bigcup_{X \in dom(\theta)} var(X\theta)$.

As usual while describing semantics of non-strict languages, we enlarge the signature with a new 0-ary constructor symbol $\bot$, which can be used to build the sets $Expr_\bot, Pat_\bot, PSubst_\bot$ of *partial* expressions, patterns and p-substitutions resp. Partial expressions are ordered by the *approximation* ordering $\sqsubseteq$ defined as the least partial ordering satisfying $\bot \sqsubseteq e$ and $e \sqsubseteq e' \Rightarrow \mathcal{C}[e] \sqsubseteq \mathcal{C}[e']$ for all $e, e' \in Exp_\bot, \mathcal{C} \in Cntxt$. This partial ordering can be extended to substitutions: given $\theta, \sigma \in Subst_\bot$ we say $\theta \sqsubseteq \sigma$ if $X\theta \sqsubseteq X\sigma$ for all $X \in \mathcal{V}$.

A *HOCRWL*-program (or simply a *program*) consists of one or more *program rules* for each $f \in FS^n$, having the form $f\ t_1 \ldots t_n \to r$ where $(t_1, \ldots, t_n)$ is a linear (i.e. variables occur only once) tuple of (maybe HO) patterns and r is any expression. Notice that confluence or termination is not required, and that r may have variables not occurring in $f\ t_1 \ldots t_n$ (we write $vExtra(R)$ for such variables in a rule R). The original *HOCRWL* logic considered also *joinability* conditions in rules to achieve a better treatment of strict equality as built-in, which is a subject orthogonal to the aims of this paper. Therefore, we consider only unconditional rules.

Some related languages, like Curry, do not allow HO-patterns in left-hand sides of function definitions. We remark that all the notions and results in the paper are applicable to programs with this restriction and we stress the fact that Example 1 is one of them.

Given a program $\mathcal{P}$, the set of its rule instances is $[\mathcal{P}] = \{(l \to r)\theta \mid (l \to r) \in \mathcal{P}, \theta \in PSubst\}$. The set $[\mathcal{P}]_\bot$ is defined similarly replacing $PSubst$ by $PSubst_\bot$. To require $\theta \in PSubst_{(\bot)}$ instead of $\theta \in Subst_{(\bot)}$ is essential to achieve call-time choice in the next sections.

2.2 The *HOCRWL* proof calculus [7]

The semantics of a program $\mathcal{P}$ is determined in *HOCRWL* by means of a proof calculus able to derive reduction statements of the form $e \twoheadrightarrow t$, with $e \in Exp_\bot$ and $t \in Pat_\bot$, meaning informally that t is (or approximates to) a possible value of e, obtained by evaluation of e using $\mathcal{P}$ under call-time choice. Besides this logical semantics, *HOCRWL* programs come in [7] with a model-theoretic semantics based on applicative algebras, with existence of a least Herbrand model. We will not use this aspect of the semantics here.

The *HOCRWL*-proof calculus is presented in Fig. 1. We write $\mathcal{P} \vdash_{HOCRWL} e \twoheadrightarrow t$ to express that $e \twoheadrightarrow t$ is derivable in that calculus using the program $\mathcal{P}$.

The *HOCRWL-denotation* of an expression $e \in Exp_\bot$ is defined as $[\![e]\!]^{\mathcal{P}}_{HOCRWL} = \{t \in Pat_\bot \mid \mathcal{P} \vdash_{HOCRWL} e \twoheadrightarrow t\}$. $\mathcal{P}$ and *HOCRWL* are frequently omitted in those notations.

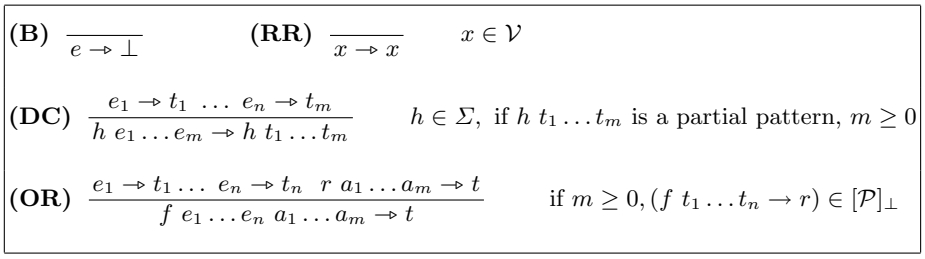


Fig. 1. (*HOCRWL*-calculus)

In Example 1 we have $[\![f double\ f\ 0]\!] = \{0, s\ (s\ 0), \bot, s\ \ \bot, s\ (s\ \ \bot)\}$ and $[\![f double\ f'\ 0]\!] = \{0, s\ 0, s\ (s\ 0), \bot, s\ \ \bot, s\ (s\ \ \bot)\}$.

We will use the following (new) result stating an important compositionality property of the semantics of *HOCRWL*-expressions: the semantics of a whole expression depends only on the semantics of its constituents, in a particular form reflecting the idea of call-time choice. The second part of the theorem is a technical result, needed in some proofs, concerning the size of the involved derivations.

Theorem 1 (Compositionality of *HOCRWL* semantics).

(i) $[\![\mathcal{C}[e]]\!] = \bigcup_{t \in [\![e]\!]} [\![\mathcal{C}[t]]\!]$*, for any program* $\mathcal{P}$ *and expression* $e \in Exp_\bot$*.*
In other terms, $\mathcal{C}[e] \twoheadrightarrow t \Leftrightarrow \exists s.(e \twoheadrightarrow s \wedge \mathcal{C}[s] \twoheadrightarrow t)$*.*

(ii) In the ($\Rightarrow$) *part of (i), if* $t \neq \bot, \mathcal{C} \neq [\ ]$ *and the derivation of* $\mathcal{C}[e] \twoheadrightarrow t$ *has size* K*, then the derivations of* $e \twoheadrightarrow s$ *and* $\mathcal{C}[s] \twoheadrightarrow t$ *can be chosen with sizes* $< K$ *and* $\leq K$ *respectively.*

3 Higher order *let*-rewriting

To express sharing, as is required for call-time choice, we enhance the syntax of expressions (and contexts) with a *let* construct for local bindings, in the spirit of [5, 21, 19]: $LExp \ni e ::= X \mid h \mid e_1\ e_2 \mid let\ X = e_1\ in\ e_2$
$Cntxt \ni \mathcal{C} ::= [\ ] \mid \mathcal{C}\ e \mid e\ \mathcal{C} \mid let\ X = \mathcal{C}\ in\ e \mid let\ X = e\ in\ \mathcal{C}$
We consider expressions $let\ X = e_1\ in\ e_2$ as passive and rigid. The sets $FV(e)$ and $BV(e)$ of free and bound variables resp. of a *let*-expression e are defined as:

$$FV(X) = \{X\};\ FV(h\ \overline{e}) = \bigcup_{e_i \in \overline{e}} FV(e_i);$$
$$FV(let\ X = e_1\ in\ e_2) = FV(e_1) \cup (FV(e_2) \backslash \{X\});$$
$$BV(X) = \emptyset;\ BV(h(\overline{e})) = \bigcup_{e_i \in \overline{e}} BV(e_i);$$
$$BV(let\ X = e_1\ in\ e_2) = BV(e_1) \cup BV(e_2) \cup \{X\}$$

Notice that with the given definition of $FV(let\ X = e_1\ in\ e_2)$ recursive *let*-bindings are not allowed since the possible occurrences of X in e_1 are not considered as bound and therefore refer to a 'different' X. We assume appropriate renamings of bound variables ensuring that bound and free variables are kept distinct, and that whenever θ is applied to $e \in LExp$, $BV(e) \cap (dom(\theta) \cup vRan(\theta)) = \emptyset$, so that $(let\ X = e_1\ in\ e_2)\theta = let\ X = e_1\theta\ in\ e_2\theta$ and $(\mathcal{C}[e])\theta = \mathcal{C}\theta[e\theta]$.

The *shell* of an expression, written as $|e|$, is a pattern containing the 'stable' outer information of e, not to be destroyed by reduction:

$$|X\ e_1 \dots e_m| = \begin{cases} X & \text{if } m = 0 \\ \bot & \text{if } m > 0 \end{cases}$$

$$|h\ e_1 \dots e_m| = \begin{cases} h\ |e_1| \dots |e_m| & \text{if } (h \in CS^n, m \leq n) \text{ or } (h \in FS^n, m < n) \\ \bot & \text{otherwise (junk or active expression)} \end{cases}$$

$$|(let\ X = e_1\ in\ e_2)\ a_1 \dots a_m| = |(e_2[X/e_1])\ a_1 \dots a_m|$$

Notice that in FO [19] we defined $|(let\ X = e_1\ in\ e_2)| = |e_2|[X/|e_1|]$. This would lose information in the HO case: for instance, $|let\ X = s\ in\ X\ 0|$ would be $\bot$, instead of the more accurate $s\ 0$ given by the definition above.

The $HOCRWL_{let}$ proof calculus for proving statements $e \twoheadrightarrow t$ ($e \in LExp_\bot, t \in Pat_\bot$) results from adding to Fig. 1 the rule:

$$\textbf{(Let)} \quad \frac{e_1 \twoheadrightarrow t_1 \quad (e_2[X/t_1])\ a_1 \dots a_m \twoheadrightarrow t}{(let\ X = e_1\ in\ e_2)\ a_1 \dots a_m \twoheadrightarrow t} \qquad (m \geq 0)$$

It is easy to see that for programs and expressions without *let*s both calculi coincide, giving $[\![e]\!]_{HOCRWL} = [\![e]\!]_{HOCRWL_{let}}$, and then we write simply $[\![e]\!]$.

Theorem 1 does not hold as it is for *let*-expressions (assume, for instance, the program rule $f\ 0 = 1$ and take $e \equiv f\ X$, $\mathcal{C} \equiv let\ X{=}0\ in\ [\ ]$). However, a more limited form of compositionality will suffice to our needs:

Theorem 2 (Weak compositionality of $HOCRWL_{let}$ semantics). *For any $\mathcal{P}$ and $e, e' \in LExp_\bot$: $[\![C\ [e]]\!] = \bigcup_{t \in [\![e]\!]} [\![C\ [t]]\!]$, if $BV(C) \cap FV(e) = \emptyset$.*

As a consequence, (i) $[\![e\ e']\!] = \bigcup_{t \in [\![e]\!]} [\![t\ e']\!]$ *(ii)* $[\![e\ e']\!] = \bigcup_{t \in [\![e']\!]} [\![e\ t]\!]$

(iii) $[\![let\ X = e\ in\ e']\!] = \bigcup_{t \in [\![e]\!]} [\![e'[X/t]]\!]$

3.1 Rewriting with local bindings

Figure 2 defines the *HOlet*-rewriting relation $\rightarrow^l$. Rule *(Fapp)* uses a program rule to reduce a function application, but only when the arguments are already patterns, otherwise call-time choice would be violated. Non-pattern arguments of applications are moved to local bindings by *(LetIn)*. Local bindings of patterns to variables are applied in *(Bind)*, since in this case copying is harmless. *(Elim)* erases useless bindings. *(Flat)* and *(LetAp)* manage local bindings; they are needed to avoid some reductions to get stuck. Notice that with the variable convention, the condition $Y \notin FV(e_3)$ in *(Flat)* and *(LetAp)* would not be needed; we have written it in order to keep the rules independent of the convention. Finally, any of these rules can be applied to any subexpression by *(Contx)*.

It includes an additional technical condition to avoid undesired variable captures when *(Fapp)* was applied inside a surrounding context and the used program rule has extra variables. If, for instance, a program rule is $f \to Y$, the rule *(Contxt)* avoids the step *let X=0 in* $f \to^l$ *let X=0 in X* and also the step *let X=f in X* $\to^l$ *let X=X in X*.

(Fapp) $f\ t_1 \dots t_n \to^l r$, if $(f\ t_1 \dots t_n \to r) \in [\mathcal{P}]$

(LetIn) $e_1\ e_2 \to^l let\ X = e_2\ in\ e_1\ X$ (X fresh), if e_2 is an active expression, variable application, junk or *let* rooted expression.

(Bind) $let\ X = t\ in\ e \to^l e[X/t]$, if $t \in Pat$

(Elim) $let\ X = e_1\ in\ e_2 \to^l e_2$, if $X \notin FV(e_2)$

(Flat) $let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3 \to^l let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3)$ if $Y \notin FV(e_3)$

(LetAp) $(let\ X = e_1\ in\ e_2)\ e_3 \to^l let\ X = e_1\ in\ e_2\ e_3$, if $X \notin FV(e_3)$

(Contx) $\mathcal{C}[e] \to^l \mathcal{C}[e']$, if $\mathcal{C} \neq [\ ]$, $e \to^l e'$ using any of the previous rules, and in case $e \to^l e'$ is a (Fapp) step using $(f\ \overline{p} \to r)\theta \in [\mathcal{P}]$ then $vRan(\theta|_{\setminus var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$.

Fig. 2. Higher order *let*-rewriting relation $\to^l$

The following derivation corresponds to Example 1:

fdouble f 0 $\to^l_{\{LetIn,Cntx\}}$ *(let F=f in fdouble F) 0*
$\to^l_{LetAp}$ *let F=f in fdouble F 0* $\to^l_{\{Fapp,Cntx\}}$ *let F=f in fadd F F 0*
$\to^l_{\{Fapp,Cntx\}}$ *let F=f in F 0 + F 0*
$\to^l_{\{Fapp,Cntx\}}$ *let F=g in F 0 + F 0* $\to^l_{Bind}$ *g 0 + g 0* $\to^{l^*}$ *0*

Notice that the fist step is justified because f is active. In contrast, since f' is a pattern, a derivation for *fdouble f' 0* could proceed as follows:

fdouble f' 0 $\to^l$ *fadd f' f' 0* $\to^l$ *f' 0 + f' 0* $\to^{l^*}$ *f 0 + f 0* $\to^{l^*}$ *g 0 + h 0* $\to^{l^*}$ *s 0*

The rules of $\to^l$ have been carefully tuned up to ensure that program rules are the only possible source of non-termination, as ensured by the following result.

Proposition 1. *The relation* $\to^l_{\setminus Fapp}$ *defined by the rules of Fig. 2 except* (Fapp) *is terminating.*

This is a natural requirement. However, at some point we will find useful to consider the more liberal relation $\to^L$ obtained replacing *(LetIn)* by:

(LetIn') $e_1\ e_2 \to^L let\ X = e_2\ in\ e_1\ X$ (X fresh)

which is less restrictive (then $\to^l \subseteq \to^L$). However $\to^L_{\setminus Fapp}$ becomes non-terminating, as shown by: $s\ 0 \to^l_{LetIn'} let\ X = 0\ in\ s\ X \to^l_{Bind} s\ 0 \to^l \dots$

3.2 Adequacy of *HOlet*-rewriting to *HOCRWL*

We compare here $\rightarrow^l$ to *HOCRWL*-derivability $\twoheadrightarrow$, proving that essentially $\rightarrow^l$ gives no more (*soundness*) and no less (*completeness*) results than $\twoheadrightarrow$.

As in [19], the following notion is useful to establish soundness:

Definition 1 (Hypersemantics).

(i) The hypersemantics *of an expression $e \in LExp_\bot$, written as $[\![e]\!]$, is a mapping $[\![e]\!] : PSubst_\bot \longrightarrow \mathcal{P}(Pat_\bot)$ defined by $[\![e]\!](\theta) = [\![e\theta]\!]$.*
(ii) Hypersemantics of expressions are ordered as follows:

$$[\![e_1]\!] \Subset [\![e_2]\!] \text{ iff } [\![e_1\theta]\!] \subseteq [\![e_2\theta]\!],\ \forall \theta \in PSubst_\bot$$

The main reason for introducing hypersemantics is that it enjoys the following nice monotonicity-under-contexts property, while $[\![\_]\!]$ does not:

Lemma 1 (Monotonicity of hypersemantics).
$[\![e]\!] \Subset [\![e']\!]$ implies $[\![\mathcal{C}[e]]\!] \Subset [\![\mathcal{C}[e']]\!]$, for any $e, e' \in LExp_\bot$, $\mathcal{C} \in Cntxt$.

Monotonicity under contexts is the key for our next result, stating that hypersemantics does not grow under *HOlet*-rewriting steps:

Lemma 2 (One-Step Hyper-Soundness of *HOlet*-rewriting).
$e \rightarrow^l e'$ implies $[\![e']\!] \Subset [\![e]\!]$, for any $e, e' \in LExp$.

Notice that $\Subset$ cannot be replaced here by $=$, due to non-determinism.

Lemma 2, together with the easy observation that $[\![e_1]\!] \Subset [\![e_2]\!]$ implies $[\![e_1]\!] \subseteq [\![e_2]\!]$ (just take $\theta = \epsilon$) and an obvious induction over derivation lengths, leads to our main correctness result for $\rightarrow^l$:

Theorem 3 (Soundness of *HOlet*-rewriting). *Let $\mathcal{P}$ be a program, $e, e' \in LExp$. Then: (i) $e \rightarrow^{l^*} e'$ implies $[\![e']\!] \subseteq [\![e]\!]$, and therefore $e \twoheadrightarrow |e'|$*
(ii) $e \rightarrow^{l^*} t$ implies $e \twoheadrightarrow t$, for any $t \in Pat$.

The proof of this result can be easily extended to the larger relation $\rightarrow^L$ (the one which uses (LetIn') instead of (LetIn)).

Regarding completeness of *let*-rewriting, a key in the FO case was the *peeling lemma* ([19], Lemma 7), a technical result giving a kind of standard form in which the implicit or explicit sharing information contained in $e \in Exp$ can be expressed. It is not obvious how to proceed in the HO case, since straightforward generalizations of the FO peeling lemma turn out to be false. However, we have found that the following weak HO version is enough for our purposes:

Lemma 3 (Weak peeling lemma). *Let $h\ e_1 \ldots e_m \in Exp$ with $h \in \Sigma^n$ (n and m can be different). Then $h\ e_1 \ldots e_m \rightarrow^{l^*} let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m$, for some $t_1, \ldots, t_m \in Pat, \overline{a} \subseteq Exp$ such that $|\overline{a}| = \overline{\bot}, t_i \equiv e_i$ for every $e_i \in Pat$. Besides, in this derivation the rule (Fapp) is not applied.*

With this result and some monotonicity properties of *HOCRWL*-derivability, we can prove a very technical but strong completeness result for $\rightarrow^l$ wrt $\twoheadrightarrow$:

Lemma 4 (Completeness lemma for *HOlet*-rewriting). *For any program $\mathcal{P}$, $e \in Exp$ and $t \in Pat_\bot$ with $t \neq \bot$, the following holds: $\mathcal{P} \vdash_{HOCRWL} e \twoheadrightarrow t$ implies $e \rightarrow^{l^*} let\ \overline{X = a}\ in\ t'$, for some $t' \in Pat$ and $\overline{a} \subseteq Exp$ in such a way that $t \sqsubseteq |let\ \overline{X = a}\ in\ t'|$ and $|a_i| = \bot$ for all $a_i \in \overline{a}$. As a consequence, $t \sqsubseteq t'[\overline{X/\ \bot}]$.*

The condition $t \neq \bot$ is needed, as can be seen just taking $\mathcal{P} = \{f \rightarrow f\}$, $e \equiv f$ and $t \equiv \bot$.

From Lemma 4 we can obtain our main completeness result for $\rightarrow^l$:

Theorem 4 (Completeness of *HOlet*-rewriting). *Let $\mathcal{P}$ be a program, $e \in Exp$, and $t \in Pat_\bot$. Then:*
(i) $\mathcal{P} \vdash_{HOCRWL} e \twoheadrightarrow t$ implies $e \rightarrow^{l^*} e'$, for some $e' \in LExp$ such that $t \sqsubseteq |e'|$.
(ii) If in addition $t \in Pat$, then $e \rightarrow^{l^*} t$.

Joining together the last parts of Theorems 3 and 4, we obtain a strong equivalence result for $\rightarrow^l$ and $\twoheadrightarrow$:

Theorem 5 (Equivalence of *HOlet*-rewriting and *HOCRWL*).
$\mathcal{P} \vdash_{HOCRWL} e \twoheadrightarrow t$ iff $e \rightarrow^{l^*} t$, for any $\mathcal{P}$, $e \in Exp$, and $t \in Pat$.

This justifies our claim that $\rightarrow^l$ is truly the reduction face of *HOCRWL*-semantics.

4 Higher order *let*-narrowing

For some FLP computations rewriting is not enough, and must be lifted to some kind of *narrowing*; this happens when the expression being reduced contains variables for which different bindings might produce different evaluation results. Narrowing is an old subject in the fields of theorem proving and declarative programming. Since classical rewriting is not correct for call-time choice, classical narrowing cannot be either (because rewriting is a particular case of narrowing). In [18] we proposed a notion of narrowing adequate to FO *let*-rewriting, and now we extend it to HO. As happens in [7, 4], *HOlet*-narrowing may bind variables to HO-patterns.

Figure 3 contains the rules for the one-step *HOlet*-narrowing relation $e \leadsto^l_\theta e'$, expressing that e is narrowed to e' producing the substitution $\theta \in PSubst$. In *(X)* we collect those cases of *HOlet*-rewriting corresponding also to narrowing steps with empty substitution. *(Narr)* is the proper rule of narrowing for function application; it may produce HO bindings if the used program rule has HO patterns. Notice that, for the sake of generality, we do not require that θ is a mgu. *(VAct)* and *(VBind)* are rules producing HO bindings for flexible expressions (or subexpressions, in the case of *(VBind)*). We have preferred this pair of rules instead of the rule

$$\textbf{(VNarr)} \quad X\ e \leadsto^L_{[X/t]} t\ (e[X/t]), \text{ for any } t \in Pat$$

which is simpler, but also 'wilder' because it creates a larger search space. Finally, (Contxt) is a contextual rule where, as in [18], it is crucial to protect bound variables from narrowing (condition (i)) and to avoid variable capture (condition (ii), automatically fulfilled if mgu's are used in *(Narr)* and *(VAct)*, and fresh *shallow* patterns –i.e., of the form $h\ X_1 \dots X_n$– in *(VBind)*).

(X) $e \leadsto^l_\epsilon e'$ if $e \to^l e'$ using $\boldsymbol{X} \in \{Elim, Bind, Flat, LetIn, LetAp\}$ in Figure 2.

(Narr) $f\ \bar{t} \leadsto^l_\theta r\theta$, for any fresh variant $(f\ \bar{p} \to r) \in \mathcal{P}$ and $\theta \in PSubst$ such that $f\ \bar{t}\theta \equiv f\ \bar{p}\theta$.

(VAct) $X\ t_1 \dots t_k \leadsto^l_\theta r\theta$, if $k > 0$, for any fresh variant $(f\ \bar{p} \to r) \in \mathcal{P}$ and $\theta \in PSubst$ such that $(X\ t_1 \dots t_k)\theta \equiv f\ \bar{p}\theta$.

(VBind) $let\ X = e_1\ in\ e_2 \leadsto^l_\theta e_2\theta[X/e_1\theta]$, if $e_1 \notin Pat$, for any $\theta \in PSubst$ that makes $e_1\theta \in Pat$, provided that $X \notin (dom(\theta) \cup vRan(\theta))$.

(Contx) $\mathcal{C}[e] \leadsto^l_\theta \mathcal{C}\theta[e']$ for $\mathcal{C} \neq [\,]$, if $e \leadsto^l_\theta e'$ by any of the previous rules, and the following conditions hold:

i) $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$

ii) • If the step is *(Narr)* or *(VAct)* using $(f\ \bar{p} \to r) \in \mathcal{P}$, then $vRan(\theta|_{\setminus var(\bar{p})}) \cap BV(\mathcal{C}) = \emptyset$
• If the step is *(VBind)* then $vRan(\theta) \cap BV(\mathcal{C}) = \emptyset$

Fig. 3. Higher order *let*-narrowing calculus $\leadsto^l$

Taking Example 1, a narrowing derivation for *fdouble F 0* would start with some *(X)* 'rewriting' steps:

$fdouble\ F\ 0 \leadsto^l_\epsilon fadd\ F\ F\ 0 \leadsto^l_\epsilon F\ 0 + F\ 0 \leadsto^l_\epsilon let\ X{=}F\ 0\ in\ X + F\ 0$

At this point, notice first that we cannot narrow on X, because it is a bound variable. Instead, we can apply *(VAct+Contx)*:

$$let\ X{=}F\ 0\ in\ X + F\ 0 \leadsto^l_{\{F/g\}} let\ X{=}0\ in\ X + g\ 0 \leadsto^{l*}_\epsilon 0$$

Other similar derivations using *(VAct+Contx)* would bind F to h (with final result *s (s 0)*), or to f' (with possible results *0, s 0, s (s 0)*). Notice that the binding X/f is not legal, since f is not a pattern.

Alternatively we could have applied *(VBind)*, obtaining:

$$let\ X{=}F\ 0\ in\ X + F\ 0 \leadsto^l_{\{F/s\}} s\ 0 + s\ 0 \leadsto^{l*}_\epsilon s\ (s\ 0)$$

We remark that, in our untyped framework, other 'ill-typed' bindings could be tried, like $F/fadd\ 0$ or $F/fdouble$. This is a symptom of known problems [4, 8] of the interaction with types of the intensional view of HO, that are partially alleviated in [4] by a typed version of a FO translation (see Sect. 6), but in general require (see [8]) bringing types to computations, a problem yet not well solved in practice. All these type-related issues are out of the scope of the paper.

A basic fact about completeness of *let*-narrowing in the FO case was that $e \leadsto^{l*}_\theta e'$ implied $e\theta \to^{l*} e'$, $\forall \theta \in CSubst$, which is closely related to the fact that FO *let*-rewriting is closed under c-substitutions. None of both facts hold with

HO $\rightsquigarrow^l$, $\rightarrow^l$ and $\theta \in PSubst$: consider for instance $e \equiv s\ (Y\ 0) \rightarrow^l let\ X = Y\ 0\ in\ s\ X \equiv e'$ and $\theta = [Y/s]$, for which $e\theta \equiv s\ (s\ 0) \not\rightarrow^l let\ X = s\ 0\ in\ s\ X \equiv e'\theta$. Similarly, we have $e \equiv s\ (Y\ 0) \rightsquigarrow^L_\epsilon let\ X = Y\ 0\ in\ s\ X \rightsquigarrow^L_{[Y/s]} let\ X = s\ 0\ in\ s\ X \equiv e'$, but $e\theta \equiv s\ (s\ 0) \not\rightarrow^l e'$.

At this point the relation $\rightarrow^L$ of Sect. 3 becomes useful, because we have:

Lemma 5 (Closedness of $\rightarrow^L$ under $PSubst$). *For every $e, e' \in LExp, \theta \in PSubst$, $e \rightarrow^{L^*} e'$ implies $e\theta \rightarrow^{L^*} e'\theta$.*

Now we can prove soundness of HO *let*-narrowing wrt. $\rightarrow^L$:

Theorem 6 (Soundness or $\rightsquigarrow^l$ wrt $\rightarrow^L$). *For any $e, e' \in LExp$, $e \rightsquigarrow^{l^*}_\theta e'$ implies $e\theta \rightarrow^{L^*} e'$.*

And now, taking into account Th. 3 (which holds also for $\rightarrow^L$), we get:

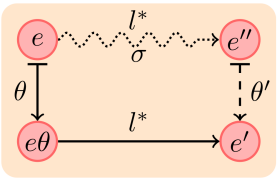
Theorem 7 (Soundness of *let*-narrowing). *For any $e, e' \in LExp$, $t \in Pat$:*
a) If $e \rightsquigarrow^{l^*}_\theta e'$ then $[\![e']\!] \subseteq [\![e\theta]\!]$ *b) If $e \rightsquigarrow^{l^*}_\theta t$ then $e\theta \rightarrow^{l^*} t$*

Regarding completeness, the following lemma shows how we can lift any $\rightarrow^l$ derivation to a $\rightsquigarrow^l$ derivation. This is surely the most involved result in the paper.

Lemma 6 (Lifting lemma for *HOlet*-rewriting). *Let $e, e' \in LExp$ such that $e\theta \rightarrow^{l^*} e'$ for some $\theta \in PSubst$, and let $\mathcal{W}, \mathcal{B} \subseteq \mathcal{V}$ with $dom(\theta) \cup FV(e) \subseteq \mathcal{W}$, $BV(e) \subseteq \mathcal{B}$ and $(dom(\theta) \cup vRan(\theta)) \cap \mathcal{B} = \emptyset$, and for each instance of a program rule $R\gamma \in [\mathcal{P}]$ used in an (Fapp) step of $e\theta \rightarrow^{l^*} e'$ then $vRan(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$. Then there exist a derivation $e \rightsquigarrow^{l^*}_\sigma e''$ and $\theta' \in PSubst$ such that:*

$$\textit{(i)}\ e''\theta' = e' \qquad \textit{(ii)}\ \sigma\theta' = \theta[\mathcal{W}] \qquad \textit{(iii)}\ (dom(\theta') \cup vRan(\theta')) \cap \mathcal{B} = \emptyset$$

Besides, the HOlet*-narrowing derivation can be chosen to use mgu's at each* **(Narr)** *or* **(VAct)** *step, and fresh shallow patterns in the range for each* **(VBind)** *step. Graphically:*



With the aid of this lemma we can reach our completeness result for $\rightsquigarrow^l$:

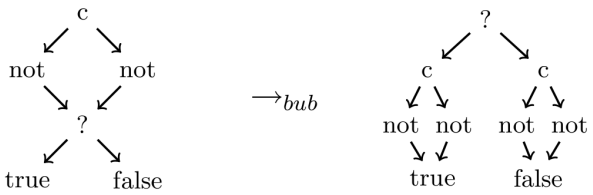
Theorem 8 (Completeness of *HOlet*-narrowing wrt. *HOlet*-rewriting). *Let $e, e' \in LExp$ and $\theta \in PSubst$. If $e\theta \rightarrow^{l^*} e'$, then there exist a* HOlet*-narrowing derivation $e \rightsquigarrow^{l^*}_\sigma e''$ and $\theta' \in PSubst$ such that $e''\theta' \equiv e'$ and $\sigma\theta' = \theta[FV(e)]$.*

5 A case of study: correctness of bubbling

Having equivalent notions of semantics and reduction allows to reason interchangeably at the rewriting and the semantic levels. We demonstrate the power of such technique by a case study where *let*-rewriting provides a good level of abstraction to formulate a new operational rule (*bubbling*), while the semantic point of view is appropriate for proving its correctness.

Bubbling, proposed in [3], is an operational rule devised to improve the efficiency of functional logic computations. Its correctness was formally studied in [2] in the framework of a variant [6] of term graph rewriting.

The idea of bubbling is to concentrate all non-determinism of a system into a *choice* operation ? defined by the rules $X\ ?\ Y \to X$ *and* $X\ ?\ Y \to Y$, and to lift applications of ? out of a surrounding context, as illustrated by the following graph transformation taken from [2]:



As it is shown in [3], bubbling can be implemented in such a way that many functional logic programs become more efficient, but we will not deal with these issues here.

Due to the technical particularities of term graph rewriting, not only the proof of correctness, but even the definition of bubbling in [3, 2] are involved and need subtle care concerning the appropriate contexts over which choices can be bubbled. In contrast, bubbling can be expressed within our framework (moreover, generalized to HO) in a remarkably easy and abstract way as a new rewriting rule: **(Bub)** $\mathcal{C}[e_1?e_2] \to^{bub} \mathcal{C}[e_1]?\mathcal{C}[e_2]$, for $e_1, e_2 \in LExp$

With this rule, the bubbling step corresponding to the graph transformation of the example above is: *let X = true ? false in c (not X) (not X)* $\to^{bub}$ *let X = true in c (not X) (not X) ? let X = false in c (not X) (not X)*

Notice that the effect of this bubbling step is not a shortening of any existing *HOlet*-rewriting derivation; bubbling is indeed a genuine new rule, the correctness of which must be therefore subject of proof. Call-time choice is essential, since bubbling is not correct with respect to run-time choice: in Example 1, *fdouble (g?h) 0* can be reduced with run-time choice to $0, 1$ or 2, while *fdouble g 0 ? fdouble h 0* leads only to 0 and 2.

The fact that bubbling preserves $HOCRWL_{let}$-semantics has a simple formulation:

Theorem 9 (Correctness of bubbling). *If* $e \to^{bub} e'$, *then* $[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$. *In other terms,* $[\![\mathcal{C}[e_1?e_2]]\!] = [\![\mathcal{C}[e_1]?\mathcal{C}[e_2]]\!]$ $(= [\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!])$, *for any* $e_1, e_2 \in LExp$ *and context* $\mathcal{C}$.

From this and the equivalence results of Sect. 3 we obtain as immediate corollary the correctness of bubbling in terms of rewriting:

Corollary 1. $e \to_l^* t \Leftrightarrow e\ (\to_l \cup \to_{bub})^*\ t$

It is interesting to observe that most of the proof of Th. 9 consists of direct calculations with denotation of expressions, in the form of chains of equalities of denotations, justified by general properties of the semantics like Th. 1. We find this methodology quite appealing and for this reason we include (a part of) the proof.

Proof (For Theorem 9, Correctness of bubbling). The proof uses the following easy (not proved here) lemma about semantics of ?, which justifies also the equation $[\![\mathcal{C}[e_1]?\mathcal{C}[e_2]]\!] = [\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!]$ stated in the Theor. 9.

Lemma 7. $[\![e_1?e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$, *for any* $e_1, e_2 \in LExp_\perp$.

Now, we reason by induction on the number k of *let*'s occurring in $\mathcal{C}[e_1?e_2]$.

- $k = 0$: Since there is no *let* in $e_1?e_2$, we can apply Theor. 1 to obtain:

$$
\begin{array}{ll}
[\![\mathcal{C}[e_1?e_2]]\!] & = \textit{(by Theor. 1)} \\
\bigcup_{t\in[\![e_1?e_2]\!]}[\![\mathcal{C}[t]]\!] & = \textit{(by Lemma 17)} \\
\bigcup_{t\in([\![\mathcal{C}[e_1]]\!]\ \cup\ [\![\mathcal{C}[e_2]]\!])}[\![\mathcal{C}[t]]\!] & = \textit{(set operations)} \\
\bigcup_{t\in[\![\mathcal{C}[e_1]]\!]}[\![\mathcal{C}[t]]\!]\ \cup\ \bigcup_{t\in[\![\mathcal{C}[e_2]]\!]}[\![\mathcal{C}[t]]\!] & = \textit{(by Theor. 1)} \\
[\![\mathcal{C}[e_1]]\!]\ \cup\ [\![\mathcal{C}[e_2]]\!] & = \textit{(by Lemma 17)} \\
[\![\mathcal{C}[e_1]\ ?\ \mathcal{C}[e_2]]\!] &
\end{array}
$$

- $k > 0$: We reason by induction on the structure of $\mathcal{C}$. The most interesting case is that of let bindings:

– $\mathcal{C} \equiv \textit{let } x = e \textit{ in } \mathcal{C}'$: then

$$
\begin{array}{ll}
[\![\mathcal{C}[e_1?e_2]]\!] & = \\
[\![\textit{let } x{=}e \textit{ in } \mathcal{C}'[e_1?e_2]]\!] & = \textit{(by Theor. 2,}\sigma \equiv \{x/t\}\textit{)} \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'[e_1?e_2]\sigma]\!] & = \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_1\sigma?e_2\sigma]]\!] & = \textit{(by IH on k, that decreases)} \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_1\sigma]?\mathcal{C}'\sigma[e_2\sigma]]\!] & = \textit{(by Lemma 17)} \\
\bigcup_{t\in[\![e]\!]}([\![\mathcal{C}'\sigma[e_1\sigma]]\!]\ \cup\ [\![\mathcal{C}'\sigma[e_2\sigma]]\!]) & = \textit{(set operations)} \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_1\sigma]]\!]\ \cup\ \bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_2\sigma]]\!] & = \textit{(by Theor. 2)} \\
[\![\textit{let } x{=}e \textit{ in } \mathcal{C}'[e_1]]\!]\ \cup\ [\![\textit{let } x{=}e \textit{ in } \mathcal{C}'[e_2]]\!] & = \\
[\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!] & = \textit{(by Lemma 17)} \\
[\![\mathcal{C}[e_1]\ ?\ \mathcal{C}[e_2]]\!] &
\end{array}
$$

6 Translation to first order

Since [28], a common technique to implement HO features in FO settings consists in a *HO-to-FO* translation introducing data constructors to represent partial applications and a special function @ (read *apply*) for reducing application of such constructors. This has been used within the context of *FLP* in [9, 4]. Here we adapt such a transformation to our context and provide a correctness proof with

respect to the semantics of the source and object programs, given by *HOCRWL* and *CRWL* [11, 19] respectively.

Definition 2 (First order translation). *Given a HOCRWL-program* $\mathcal{P} = \{f\ \overline{p_1} \to e_1, \ldots, f\ \overline{p_m} \to e_m\}$ *built up over the signature* $\Sigma = FS \cup CS$*, its* first order translation P_{fo} *will be defined over the* **extended signature** $\Sigma_{fo} = FS_{fo} \cup CS_{fo}$ *where:*

$$FS_{fo} = FS \cup \{@\}; \quad CS_{fo} = \bigcup_{c \in CS^n, n \in \mathbb{N}} \{c_0, \ldots, c_n\} \cup \bigcup_{f \in FS^n, n \in \mathbb{N}} \{f_0, \ldots, f_{n-1}\}$$

being $@$ *a new function symbol of arity 2 and* $c_0, \ldots, c_n, f_0, \ldots, f_{n-1}$ *new symbols (with arities indicated by the sub-index). The set* $\mathcal{P}_@$ *of* $@-rules$ *is defined as:*

$$\begin{array}{ll} @(c_k(X_1, \ldots, X_k), Y) & = c_{k+1}(X_1, \ldots, X_k, Y), \textit{for each } c \in DC^n, k < n \\ @(f_k(X_1, \ldots, X_k), Y) & = f_{k+1}(X_1, \ldots, X_k, Y), \textit{for each } f \in FS^n, k+1 < n \\ @(f_{n-1}(X_1, \ldots, X_{n-1}), Y) & = f(X_1, \ldots, X_{n-1}, Y), \textit{for each } f \in FS^n \end{array}$$

The transforming function $fo : Exp_{\Sigma,\bot} \to Exp_{\Sigma_{fo},\bot}$ *is defined as:*

$$\begin{array}{lll} fo(\bot) = \bot & fo(X) = X & fo(h) = h_0, \textit{if } h \in CS \textit{ or } h \in FS^n, n > 0 \\ fo(f) = f, \textit{if } f \in FS^0 & & fo(e_1\ e_2) = @(fo(e_1), fo(e_2)) \end{array}$$

The **transformed program** *is defined as* $P_{fo} = \{f(\overline{fo(p_1)\downarrow_@}) \to fo(e_1)\downarrow_@, \ldots, f(\overline{fo(p_m)\downarrow_@}) \to fo(e_m)\downarrow_@\} \cup P_@$*, where* $e\downarrow_@$ *stands for a* **normal form** *for* e *with respect to* $@-rules$ *defined above.*

The program rules obtained by the transformation are well defined: it is easy to prove that if p is a pattern then $fo(p)\downarrow_@$ is a FO constructor term.

For the program of Example 1 we have $FS_{fo} = \{+, f, g, h, f', fadd, fdouble, @\}$ and $CS_{fo} = \{0, s_0, s, +_0, +_1, g_0, h_0, f'_0, fadd_0, fadd_1, fadd_2, fdouble_0\}$. The translated rules are:

$$\begin{array}{llllll} g(X) \to 0 & f \to g_0 & f \to h_0 & f'(X) \to @(f, X) & h(X) \to s(0) \\ fadd(F, G, X) \to @(F, X) + @(G, X) & & fdouble(F) \to fadd_2(F, F) \end{array}$$

And the rules for $@$ are:

$$\begin{array}{lll} @(+_0, X) \to +_1(X) & @(s_0, X) \to s(X) & @(h_0, X) \to h(X) \\ @(+_1(X), Y) \to X + Y & @(g_0, X) \to g(X) & @(f'_0, X) \to f'(X) \\ @(fadd_0, F) \to fadd_1(F) & @(fadd_2(F, G), X) \to fadd(F, G, X) & \\ @(fadd_1(F), G) \to fadd_2(F, G) & @(fdouble_0, F) \to fdouble(F) & \end{array}$$

The translation of the expressions to reduce in that example are:

$fo(fdouble\ f\ 0)\downarrow_@ = @(fdouble(f), 0)$ $\quad fo(fdouble\ f'\ 0)\downarrow_@ = @(fdouble(f'_0), 0)$

In general we cannot expect to prove a statement of the form $fo(e) \twoheadrightarrow fo(t)$ because $fo(t)$ can contain calls to the function $@$, i.e. $fo(t)$ might not be a FO constructor term. But the same statement makes sense in the form $fo(e) \twoheadrightarrow fo(t)\downarrow_@$ because $fo(t)\downarrow_@$ is a FO constructor term.

Proposition 2. $[\![fo(e)\downarrow_@]\!]^{\mathcal{P}}_{CRWL} = [\![fo(e)]\!]^{\mathcal{P}}_{CRWL}$*. Moreover* $[\![fo(e)]\!] = [\![e']\!]$ *where* e' *is any expression obtained from* e *by reducing some calls of* $@$*.*

According to this, when proving a statement $fo(e) \twoheadrightarrow t$ we can use any equivalent expression e' (in the sense of previous lemma) in the left hand side and prove $e' \twoheadrightarrow t$.

The correctness of the transformation can be stated then as follows:

Theorem 10 (Adequacy of HO-to-FO translation). *Let $\mathcal{P}$ be a program, $e \in Exp_\perp$, $t \in Pat_\perp$. Then:* $\mathcal{P} \vdash_{HOCRWL} e \twoheadrightarrow t \Leftrightarrow \mathcal{P}_{fo} \vdash_{CRWL} fo(e) \twoheadrightarrow fo(t)\downarrow_@$ *Or, in terms of* HOlet*-rewriting:* $e \rightarrow^{l^*} t \Leftrightarrow fo(e) \rightarrow^{l^*} fo(t)\downarrow_@$

7 Conclusions

Our paper addresses the broad question: *what means 'reduction' for functional logic programming?*, which had no previous satisfactory answer for the combination *HO + non-deterministic functions + call-time choice* supported by current systems in the mainstream of the field (Curry [16], Toy [20]). This leads to subtle behaviors well characterized from the point of view of a declarative semantics [7], but with no corresponding basic notion of one-step reduction. We have made a number of identifiable **contributions** in this sense:

• We propose a notion of rewriting with local bindings (*HOlet-rewriting*) suitable for a large class of HO systems (possibly non-confluent and non-terminating, allowing extra variables in right-hand sides and HO-patterns in left-hand sides).

• We have proved equivalence of *HOlet*-rewriting wrt to *HOCRWL* [7] declarative semantics. Along the way we have extended *HOCRWL* to cope with *let*s, and established new compositional properties of *HOCRWL* semantics.

• We have lifted *HOlet*-rewriting to a notion of *HOlet-narrowing* which is able to bind variables to patterns, even HO ones representing intensional descriptions of functions. We prove soundness and completeness of *HOlet*-narrowing wrt. *HOlet*-rewriting.

• We have recast within our framework the definition and proof of correctness of *bubbling*, an operational rule investigated in [3, 2] using term graph rewriting techniques. Apart from extending it to HO, this case study illustrates quite well the power of using indistinctly rewriting and/or semantic-based reasoning.

• To close the panorama, we have formally proved that *translation from HO to FO*, a technique actually used in the implementations of FLP systems, still works properly when *let*-bindings with call-time choice are considered, while previous works [9, 4] consider only deterministic functions.

The first three points have been conceived as an extension to HO of our previous work on the FO case [19, 18]. However, adapting it has not been routine; on the contrary, some results have been indeed a technical challenge.

Our wish with this work, jointly with [19, 18], is to have provided foundational pieces useful to understand how a FLP computation proceeds, serving also as suitable technical basis to address in the call-time choice context other operational issues (rewriting and narrowing strategies, residuation, program optimization, types in computations,...), all of which are lines of future work.

References

1. E. Albert, M. Hanus, F. Huch, J. Oliver, and G. Vidal. Operational semantics for declarative multi-paradigm languages. *J. of Symb. Comp.*, 40(1):795–829, 2005.
2. S. Antoy, D. Brown, and S. Chiang. On the correctness of bubbling. *Proc. RTA'06)*, 35–49. Springer LNCS 4098, 2006.
3. S. Antoy, D. Brown, and S. Chiang. Lazy context cloning for non-deterministic graph rewriting. In *Proc. Termgraph'06,*, 61–70, ENTCS 176(1), 2007.
4. S. Antoy and A. P. Tolmach. Typed higher-order narrowing without higher-order strategies. *Proc. FLOPS'99*, 335–353, Springer LNCS 1722, 1999.
5. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *Proc. POPL'95*, 233–246, 1995.
6. R. Echahed and J.-C. Janodet. Admissible graph rewriting and narrowing. *Proc. JICSLP'98*, 325 – 340. MIT Press, 1998.
7. J. González-Moreno, M. Hortalá-González, and M. Rodríguez-Artalejo. A higher order rewriting logic for functional logic programming. *Proc. ICLP'97*, 153–167. MIT Press, 1997.
8. J. González-Moreno, T. Hortalá-González, and Rodríguez-Artalejo, M. Polymorphic types in functional logic programming. *J. of Functional and Logic Programming*, volume 2001/S01, 1–71, 2001.
9. J. C. González-Moreno. A correctness proof for warren's ho into fo translation. *Proc. GULP'93*, 569–584, 1993.
10. J. C. González-Moreno, M. T. Hortalá-González, and M. Rodríguez-Artalejo. On the completeness of narrowing as the operational semantics of functional logic programming. *Proc. CSL'92*, 216–230, Springer LNCS 702, 1992.
11. J. C. González-Moreno, T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *J. of Logic Programming*, 40(1):47–87, 1999.
12. M. Hanus. The integration of functions into logic programming: From theory to practice. *J. of Logic Programming*, 19&20:583–628, 1994.
13. M. Hanus. Curry mailing list. http://www.informatik.uni-kiel.de/ curry/listarchive/0497.html, March 2007.
14. M. Hanus. Multi-paradigm declarative languages. *Proc. ICLP 2007*, 45–75. Springer LNCS 4670, 2007.
15. M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. *J. of Functional Programming*, 9(1):33–75, 1999.
16. M. Hanus (ed.). Curry: An integrated functional logic language (version 0.8.2). Available at *http://www.informatik.uni-kiel.de/~curry/report.html*, March 2006.
17. H. Hussmann. *Non-Determinism in Algebraic Specifications and Algebraic Programs.* Birkhäuser Verlag, 1993.
18. F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. Narrowing for non-determinism with call-time choice semantics. *Proc. WLP'07*, 2007.
19. F. López-Fraguas, J. Rodríguez-Hortalá, J. Sánchez-Hernández. A simple rewrite notion for call-time choice semantics. *Proc. PPDP'07*, 197–208. ACM, 2007.
20. F. López-Fraguas and J. Sánchez-Hernández. $\mathcal{TOY}$: A multiparadigm declarative system. *Proc. RTA'99*, 244–247. Springer LNCS 1631, 1999.
21. J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. *J. Funct. Program.*, 8(3):275–317, 1998.
22. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

23. K. Nakahara, A. Middeldorp, and T. Ida. A complete narrowing calculus for higher-order functional logic programming. *Proc. PLILP'95*, 97–114, LNCS 882, 1995.
24. S.L. Peyton Jones (ed.). Haskell 98 Language and Libraries. The Revised Report. Cambridge Univ. Press, 2003.
25. D. Plump. Essentials of term graph rewriting. *ENTCS* 51, 2001.
26. F. van Raamsdonk. Higher-order rewriting. In *Term Rewriting Systems*, Cambridge Univ. Press, 2003.
27. M. Rodríguez-Artalejo. Functional and constraint logic programming. *Revised Lect. of Int. Summer School CCL'99*, 202–270. Springer LNCS 2002, 2001.
28. D. H. Warren. Higher-order extensions to prolog: are they needed? *Machine Intelligence 10*, 441–454. Ellis Horwood Ltd., 1982.

A Proofs of the results

We give here the proofs of the results. The results themshelves have not been repeated. The appendix includes also many more auxiliary results. For the sake of readability we have sectioned the appendix following the structure of the paper. Many times we refer to proofs 'for the first order case'. This points to proofs in [19] or [18] (long version in `http://gpd.sip.ucm.es/fraguas/papers/longWLP07.pdf`).

2 Preliminaries: HOCRWL

Proof (For Theorem 1, Compositionality of HOCRWL semantics). We prove both implications in (i), and for the $\Rightarrow$ part of (i) we prove also (ii).
$\boxed{\Rightarrow}$ Assume $\mathcal{C}[e] \twoheadrightarrow t$.
The cases when $t = \bot$ or $\mathcal{C} = [\,]$ are trivial just taking $s = t$ (and in these cases (ii) does not apply).

For the rest of the cases, we reason by complete induction on the size K of the derivation of $\mathcal{C}[e] \twoheadrightarrow t$. We assume then that the implication is true for any size $< K$. We need in addition distinguish cases according to the shape of $\mathcal{C}[e] \twoheadrightarrow t$ and its derivation, following the rules of HOCRWL:

(RR) $\mathcal{C}[e] \twoheadrightarrow t \equiv X \twoheadrightarrow X$: this can only happen if $e = X$ and $\mathcal{C} = [\,]$

(DC) $\mathcal{C}[e] \twoheadrightarrow t \equiv h\ e_1 \dots e_n \twoheadrightarrow h\ t_1 \dots t_n$, where $h\ t_1 \dots t_n$ is a partial pattern. The derivation must take the form

$$\frac{e_1 \twoheadrightarrow t_1 \dots e_n \twoheadrightarrow t_n}{h\ e_1 \dots e_n \twoheadrightarrow h\ t_1 \dots t_n}$$

Now we split the proof in two subcases:

- $e = h\ e_1 \dots e_k$, for $k < n$ ($k = n$ corresponds to $\mathcal{C} = [\,]$). We take $s = h\ t_1 \dots t_k$, for which there are derivations $e \twoheadrightarrow s$ and $\mathcal{C}[s] \twoheadrightarrow t$ given by
$$\frac{e_1 \twoheadrightarrow t_1 \dots e_k \twoheadrightarrow t_k}{h\ e_1 \dots e_k \twoheadrightarrow h\ t_1 \dots t_k}$$
and
$$\frac{t_1 \twoheadrightarrow t_1 \dots t_k \twoheadrightarrow t_k\ e_{k+1} \twoheadrightarrow t_{k+1} \dots e_n \twoheadrightarrow t_n}{h\ t_1 \dots t_k\ e_{k+1} \dots e_n \twoheadrightarrow h\ t_1 \dots t_n}$$
with sizes $< K$ and $\leq K$ respectively[4] .
- $e_i = \mathcal{C}'[e]$, for some $1 \leq i \leq n$. Then we have $\mathcal{C}'[e] \twoheadrightarrow t_i$ with size $K' < K$. If $t_i = \bot$ or $\mathcal{C}' = [\,]$, it is enough to take $s = t_i$. Otherwise, by the HI, there exists s such that $e \twoheadrightarrow s$ with derivation of size $< K'$ (and hence $< K$) and $\mathcal{C}'[s] \twoheadrightarrow t_i$ with derivation of size $\leq K'$. Since $\mathcal{C}[s] \equiv h\ e_1 \dots \mathcal{C}'[s] \dots e_n$, we can build the following derivation of $\mathcal{C}[s] \twoheadrightarrow t$

[4] We are using here (and we will do several more times) the easy fact that for any pattern t, $t \twoheadrightarrow t$ can be proved with a derivation not larger than any other derivation $e \twoheadrightarrow t$.

$$\frac{e_1 \twoheadrightarrow t_1 \ldots \mathcal{C}'[s] \twoheadrightarrow t_i \ldots e_n \twoheadrightarrow t_n}{h\ e_1 \ldots \mathcal{C}'[s] \ldots e_n \twoheadrightarrow h\ t_1 \ldots t_i \ldots t_n}$$

of size $\leq K$.

(OR) $\mathcal{C}[e] \twoheadrightarrow t \equiv f\ e_1 \ldots e_n\ a_1 \ldots a_m \twoheadrightarrow t$ where $f \in FS^n$. The derivation must have the form

$$\frac{e_1 \twoheadrightarrow t_1 \ldots e_n \twoheadrightarrow t_n \quad r\ a_1 \ldots a_m \twoheadrightarrow t}{f\ e_1 \ldots e_n\ a_1 \ldots a_m \twoheadrightarrow t}$$

where $f\ t_1 \ldots t_n \to r\ \in [\mathcal{P}]_\perp$.

Now we split the proof in four subcases:

- $e = f\ e_1 \ldots e_k$, for $k < n$. We take $s = f\ t_1 \ldots t_k$, for which there are derivations $e \twoheadrightarrow s$ and $\mathcal{C}[s] \twoheadrightarrow t$ given by
$$\frac{e_1 \twoheadrightarrow t_1 \ldots e_k \twoheadrightarrow t_k}{f\ e_1 \ldots e_k \twoheadrightarrow h\ t_1 \ldots t_k}$$
and
$$\frac{t_1 \twoheadrightarrow t_1 \ldots t_k \twoheadrightarrow t_k\ e_{k+1} \twoheadrightarrow t_{k+1} \ldots e_n \quad r\ a_1 \ldots a_m \twoheadrightarrow t_n}{f\ t_1 \ldots t_k\ e_{k+1} \ldots e_n\ a_1 \ldots a_m \twoheadrightarrow t}$$
with sizes $< K$ and $\leq K$ respectively.
- $e = f\ e_1 \ldots e_n\ a_1 \ldots a_k$, for $0 \leq k < m$, which corresponds to $\mathcal{C} = [\ ]\ a_{k+1} \ldots a_m$ ($k = m$ would correspond to $\mathcal{C} = [\ ]$). Now, consider $e' = r\ a_1 \ldots a_k$, so that $r\ a_1 \ldots a_m = \mathcal{C}[e']$. Since $r\ a_1 \ldots a_m \twoheadrightarrow t$ has a derivation of size $K' < K$, we can apply the IH to $\mathcal{C}[e']$ (since $t \neq \perp, \mathcal{C} \neq [\ ]$) obtaining s and derivations for $e' \twoheadrightarrow s$ and $\mathcal{C}[s] \twoheadrightarrow t$ with sizes $< K'$ and $\leq K'$. We only need to show that $e \twoheadrightarrow s$ for which we can build the derivation
$$\frac{e_1 \twoheadrightarrow t_1 \ldots e_n \twoheadrightarrow t_n \quad r\ a_1 \ldots a_k \twoheadrightarrow s}{f\ e_1 \ldots e_n\ a_1 \ldots a_k \twoheadrightarrow s}$$
which has size $< K$.
- $e_i = \mathcal{C}'[e]$, for some $1 \leq i \leq n$. In this case the proof proceed in a very similar way to the second case of (DC).
- $a_i = \mathcal{C}'[e]$, for some $1 \leq i \leq m$. Consider $\mathcal{C}'' = r\ a_1 \ldots \mathcal{C}' \ldots a_m$, so that $\mathcal{C}''[e] = r\ a_1 \ldots a_i \ldots a_m$, and therefore we have a derivation of $\mathcal{C}''[e] \twoheadrightarrow t$ of size $K' < K$. By the IH, there exist s and derivations for $e \twoheadrightarrow s$ and $\mathcal{C}''[s] \twoheadrightarrow t$ os sizes $< K'$ (hence $< K$) and $\leq K'$. Now, just notice that $\mathcal{C}''[s] = r\ a_1 \ldots s \ldots a_m$ and $\mathcal{C}[s] = f\ e_1 \ldots e_n\ a_1 \ldots s \ldots a_m$, and therefore we have a derivation for $\mathcal{C}[s] \twoheadrightarrow t$ of the form
$$\frac{e_1 \twoheadrightarrow t_1 \ldots e_n \twoheadrightarrow t_n \quad r\ a_1 \ldots s \ldots a_m \twoheadrightarrow t}{f\ e_1 \ldots e_n\ a_1 \ldots s \ldots a_m \twoheadrightarrow t}$$
and with size $\leq K$.

$\boxed{\Leftarrow}$ The proof becomes almost immediate if we use the calculus HOBRC of [7], which is shown there to be equivalent to the calculus of Fig. 1 for proving statements $e \twoheadrightarrow t$. Notice however that HOBRC is able to prove also more general statements of the form $e \twoheadrightarrow e'$, for which the following simple stability can be proved by a straightforward by induction on the structure of $\mathcal{C}$:

Lemma 8. *$e \twoheadrightarrow e' \ \Rightarrow\ \mathcal{C}[e] \twoheadrightarrow \mathcal{C}[e']$, for any e, e'*

We can now proceed with the proof of $\Leftarrow$. Assume $e \twoheadrightarrow s$ and $\mathcal{C}[s] \twoheadrightarrow t$. Then, by the previous lemma $\mathcal{C}[e] \twoheadrightarrow \mathcal{C}[s]$, and by the transitivity rule of HOBRC we obtain $\mathcal{C}[e] \twoheadrightarrow t$ as desired. Notice that transitivity of $\twoheadrightarrow$ is the only rule used from HOBRC.

3 Higher order *let*-rewriting

Firstly we introduce an alternative (but equivalent) version of the $HOCRWL_{let}$ calculus that will make easier some proofs in this section and also in Sect. 6. The variant $HOCRWL'_{let}$ is presented in Figure 4.

(B) $\dfrac{}{e \twoheadrightarrow \bot}$ **(RR)** $\dfrac{}{x \twoheadrightarrow x}$ $x \in \mathcal{V}$

(DC) $\dfrac{e_1 \twoheadrightarrow t_1 \ \ldots \ e_m \twoheadrightarrow t_m}{h\ e_1 \ldots e_m \twoheadrightarrow h\ t_1 \ldots t_m}$ $h \in \Sigma$, if $h\ t_1 \ldots t_m$ is a partial pattern, $m \geq 0$

(OR) $\dfrac{e_1 \twoheadrightarrow t_1 \ldots \ e_n \twoheadrightarrow t_n \quad r \twoheadrightarrow t}{f\ e_1 \ldots e_n \twoheadrightarrow t}$ if t is a partial pattern, $(f\ t_1, \ldots, t_n \to r) \in [\mathcal{P}]_\bot$

(Let) $\dfrac{e_1 \twoheadrightarrow t_1 \quad e_2[X/t_1] \twoheadrightarrow t}{let\ X = e_1\ in\ e_2 \twoheadrightarrow t}$ if t is a partial pattern

(Ap) $\dfrac{e_1 \twoheadrightarrow t_1 \quad e_2 \twoheadrightarrow t_2 \quad t_1\ t_2 \twoheadrightarrow t}{e_1\ e_2 \twoheadrightarrow t}$

Fig. 4. $HOCRWL'_{let}$: an alternative to $HOCRWL_{let}$

With respect to the original version $HOCRWL_{let}$ introduced in Sect. 3, in this calculus the rules **(OR)** and **(Let)** are modified and a new rule **(Ap)** is added. Both presentations are equivalent in the sense that they prove exactly the same approximation statements as shows the following result.

Lemma 9 (Equivalence of $HOCRWL_{let}$ and $HOCRWL'_{let}$). *For any let-expression e we have $[\![e]\!]^{HOCRWL_{let}} = [\![e]\!]^{HOCRWL'_{let}}$.*

Proof. It is a direct application of Lemma 2. The only case in which the derivations are really different is when $e = (e_1\ e_2)$. In this case by the cited Lemma we have in $HOCRWL_{let}$:

$$t \in [\![e_1\ e_2]\!] \Leftrightarrow t \in \bigcup_{t_1 \in [\![e_1]\!]} \bigcup_{t_2 \in [\![e_2]\!]} [\![t_1\ t_2]\!] \Leftrightarrow \exists t_1 \in [\![e_1]\!], t_2 \in [\![e_2]\!].t_1\ t_2 \twoheadrightarrow t$$

But then, in $HOCRWL'_{let}$ we can build the proof:

$$\frac{e_1 \twoheadrightarrow t_1 \quad e_2 \twoheadrightarrow t_2 \quad (t_1\ t_2) \twoheadrightarrow t}{(e_1\ e_2) \twoheadrightarrow t}\ AP$$

Notice that the implications are bidirectional.

Proof (For Theorem 2, Weak compositionality of HOCRWL$_{let}$ *semantics).*

The proof becomes easier using an alternative version of *HOCRWL'*$_{let}$ introduced in Fig 4.

We must prove $C\ [e] \twoheadrightarrow t \Leftrightarrow \exists s$ such that $e \twoheadrightarrow s$ and $C\ [s] \twoheadrightarrow t$. By induction on the size of the proof for $C\ [e] \twoheadrightarrow t$. The base case only allows the proofs $C\ [e] \to \bot$, $C\ [e] \equiv X \to X$ and $C\ [e] \equiv h \to h$ with $h \in Pat$, that are clear.

Now, for the inductive step we consider the rule applied:

(DC) If the proof is:

$$\frac{e_1 \twoheadrightarrow t_1 \ldots e_n \twoheadrightarrow t_n}{h\ e_1\ \ldots\ e_n \twoheadrightarrow h\ t_1\ \ldots\ t_n}$$

and $C\ [e] = h\ e_1\ \ldots\ e_n$ we have two possible situations:

- $C\ [e] = h\ e_1\ \ldots (C'\ [e]) \ldots e_n$, i.e., e is a subexpression of some e_i. In this case, by i.h. $C'\ [e] \twoheadrightarrow t_i' \Leftrightarrow \exists s \in [\![e]\!]$ such that $C'\ [s] \twoheadrightarrow t_i'$. Then $C\ [s] = h\ e_1 \ldots (C'\ [s]) \ldots e_n$ and it is direct to build the proof for $C\ [s] \twoheadrightarrow t$.
- $C\ [e] = C'\ [h\ e_1 \ldots e_k]$ with $C' = []\ e_{k+1} \ldots e_n$ (with $k \geq 0$). We can take $s = h\ t_1 \ldots t_k \in [\![e]\!]$ and build the proof:

$$\frac{h\ t_1\ \ldots t_k \twoheadrightarrow h\ t_1\ \ldots t_k \quad e_{k+1} \twoheadrightarrow t_{k+1} \ldots e_n \twoheadrightarrow t_n}{C\ [s] = (h\ t_1\ \ldots t_k)\ e_{k+1}\ \ldots\ e_n \twoheadrightarrow h\ t_1\ \ldots t_k\ t_{k+1} \ldots t_n}$$

(OR) The proof is similar to the previous case.

(Let) It the proof is:

$$\frac{e_1 \twoheadrightarrow t_1 \quad e_2[X/t_1] \twoheadrightarrow t}{let\ X = e_1\ in\ e_2 \twoheadrightarrow t}$$

We have two possible situations:

- $C\ [e] = (let\ X = C'\ in\ e_2)\ [e]$, with $e_1 = C'\ [e]$ (e is a subexpression of e_1). By i.h. we have $e_1 \twoheadrightarrow t_1 \Leftrightarrow \exists s \in [\![e]\!]$ such that $C'\ [s] \twoheadrightarrow t_1$. Then we have:

$$\frac{C'\ [s] \twoheadrightarrow t_1 \quad e_2[X/t_1] \twoheadrightarrow t_2}{C\ [s] \equiv let\ X = C'\ [s]\ in\ e_2 \twoheadrightarrow t}$$

- $C\ [e] = (let\ X = e_1\ in\ C')\ [e]$, with $e_2 = C'\ [e]$ (e is a subexpression of e_2). We have $e_2[X/t_1] = (C'\ [e])[X/t_1] = C'[X/t_1]\ [e[X/t_1]]$, but as $X \in BV(C)$ it must be $X \notin FV(e)$, and then $e_2[X/t_1] = C'[X/t_1]\ [e]$. By i.h. $e_2[X/t_1] \equiv C'[X/t_1]\ [e] \twoheadrightarrow t_2 \Leftrightarrow \exists s \in [\![e]\!]$ such that $C'[X/t_1]\ [s] \twoheadrightarrow t$. Now we can build:

$$\frac{e_1 \twoheadrightarrow t_1 \quad (C'\ [s])[X/t_1] \equiv^{(*)} C'[X/t_1]\ [s] \twoheadrightarrow t}{C\ [s] \equiv let\ X = e_1\ in\ C'\ [s] \twoheadrightarrow t}$$

The equivalence (*) comes from: as $X \notin FV(e)$ and $e \to s$ then $X \notin var(s)$ (s can introduce fresh variables, but not X).

(Ap) It the proof is:

$$\frac{e_1 \twoheadrightarrow t_1 \quad e_2 \twoheadrightarrow t_2 \quad t_1\ t_2 \twoheadrightarrow t}{e_1\ e_2 \twoheadrightarrow t}$$

Again we have two possible situations:

- $C\ [e] = (C'\ e_2)\ [e]$ being e a subexpression of e_1. By i.h. we have $C'\ [e] \twoheadrightarrow t_1 \Leftrightarrow \exists s \in [\![e]\!]$ such that $C'\ [s] \twoheadrightarrow t_1$. It is easy to build the proof for $(C'\ e_2)\ [s] \twoheadrightarrow t$
- $C\ [e] = (e_1\ C')\ [e]$ being e a subexpression of e_2. Similar.

This ends the proof of the main part of the theorem. With respect to consequences *(i)*, *(ii)* , and *(iii)*, the first two follow easily: for *(i)*, $(e\ e') \equiv \mathcal{C}[[]e]$ where $\mathcal{C} = ([\ ]\ e')$, and we can rename bound variables in e' to avoid clashes with free variables of e. Then it suffices to apply the main part. The case *(ii)* is similar.

For (iii), we can reason again that $let\ x = e\ in\ e' \equiv \mathcal{C}[[]e]$ where $\mathcal{C} = let\ x = [\ ]\ in\ e'$, and from the main part (as before, renaming in e' if necessary) we obtain $[\![let\ x = e\ in\ e']\!] = \bigcup_{t \in [\![e]\!]} [\![let\ x = t\ in\ e']\!]$. Now, using that $let\ x = t\ in\ e'$ is not inside any context and $t \in Pat_\perp$, it is easy to prove, by reasoning directly over the HOCRWL rule for *(Let)*, that $[\![let\ x = t\ in\ e']\!] = [\![e'\sigma]\!]$, where $\sigma = \{X/t\}$.

Proof (For Proposition 1). We define for any $e \in LExp$ the size (k_1, k_2, k_3, k_4), where

$k_1 \equiv$ *number of subexpressions in* e *to which (LetAp) is applicable.*
$k_2 \equiv$ *number of subexpressions in* e *to which (LetIn) is applicable.*
$k_3 \equiv$ *number of* lets *in* e.
$k_4 \equiv$ *sum of the levels of nesting of all* let*-subexpressions in* e.

Sizes are lexicographically ordered. We prove now that application of *(LetIn), (Bind), (Elim), (Flat), (LetAp)* in any context (hence, also the application of *(Contxt)*) decreases the size, what proves termination of $\rightarrow^l_{\setminus FApp}$. The effect of each rule in the size is summarized as follows (in each case, we stop at the decreasing component):

(LetIn): $(=, <, \_, \_)$
(Bind): $(=, =, <, \_)$
(Elim): $(\leq, \leq, <, \_)$
(Flat): $(=, =, =, <)$
(LetAp): $(<, \_, \_, \_)$

Lemma 10 (Substitution lemma for *let*-expressions). *Given* $e, e' \in LExp_\perp$, $\theta \in LSubst_\perp$ *and* $X \in \mathcal{V}$ *such that* $X \notin dom(\theta)$ *and* $X \notin vRan(\theta)$, *then*

$$(e[X/e'])\theta \equiv e\theta[X/e'\theta]$$

Proof. This proof almost identical to the corresponding proof in first order, which proceeds by induction on the structure of e', just replacing the case for $e \equiv h(e_1, \ldots, e_n)$ with the following cases:

- $e \equiv h \in \Sigma$: trivial, because h is not affected by any substitution.

– $e \equiv e_1\ e_2$: Then

$$(e[X/e'])\theta \equiv (e_1\ e_2)[X/e']\theta \equiv (e_1[X/e']\theta)\ (e_2[X/e']\theta)$$
$$\equiv_{HI} (e_1\theta[X/e'\theta])\ (e_2\theta[X/e'\theta]) \equiv (e_1\ e_2)\theta[X/e'\theta] \equiv e\theta[X/e'\theta]$$

Lemma 11. *For any $\theta \in Susbst_\perp$, $\mathcal{C} \in Cntxt$ and $e \in LExp_\perp$ such that $dom(\theta) \cap BV(\mathcal{C}) = vRan(\theta) \cap BV(\mathcal{C}) = \emptyset$, we have $(\mathcal{C}[e])\theta \equiv \mathcal{C}\theta[e\theta]$.*

Proof. This proof is again very similar to the corresponding proof in first order, just replacing the case for first order application with the cases for the two kinds of higher order application context:

– $\mathcal{C} \equiv \mathcal{C}'\ a$: Then

$$(\mathcal{C}[e])\theta \equiv (\mathcal{C}'[e]\ a)\theta \equiv (\mathcal{C}'[e]\theta)\ (a\theta) \equiv_{IH} (\mathcal{C}'\theta[e\theta])\ (a\theta)$$
$$\equiv ((\mathcal{C}'\theta[])\ (a\theta))[e\theta] \equiv ((\mathcal{C}'[]\ a)\theta)[e\theta] \equiv \mathcal{C}\theta[e\theta]$$

– $\mathcal{C} \equiv a\ \mathcal{C}'$: Then

$$(\mathcal{C}[e])\theta \equiv (a\ (\mathcal{C}'[e]))\theta \equiv (a\theta)\ (\mathcal{C}'[e]\theta) \equiv_{IH} (a\theta)\ (\mathcal{C}'\theta[e\theta])$$
$$\equiv ((a\theta)\ (\mathcal{C}'\theta[]))[e\theta] \equiv ((a\ (\mathcal{C}'[]))\theta)[e\theta] \equiv \mathcal{C}\theta[e\theta]$$

Lemma 12. $\forall t \in Pat_\perp, |t| \equiv t$

Proof. A simple induction on the structure of patterns.

Lemma 13. *For any $h \in \Sigma, \sigma \in Susbt_\perp, e_1, \ldots, e_m \in LExp_\perp, m \geq 0$ we have $(h\ e_1 \ldots e_m)\sigma \equiv h\ (e_1\sigma) \ldots (e_m\sigma)$*

Proof. A simple induction over m, using left associativity of application.

Lemma 14. *For any $e \in LExp_\perp, t \in Pat_\perp$:*

a) $|let\ X = e\ in\ t| \equiv t[X/|e|]$
b) $|t[X/e]| \equiv t[X/|e|]$

Proof. a) is just a consequence of *b)*, as $|let\ X = e\ in\ t| \equiv |t[x/e]| \equiv_{b)} t[X/|e|]$. And we can prove *b)* by induction over the structure of the pattern t.

Base cases

– $t \equiv X$. $|t[X/e]| \equiv |X[X/e]| \equiv |e|| \equiv X[X/|e|] \equiv t[X/|e|]$

– $t \equiv Y \not\equiv X$. $|t[X/e]| \equiv |Y[X/e]| \equiv |Y| \equiv Y \equiv Y[X/|e|] \equiv t[X/|e|]$

– $t \equiv h \in CS^m, m \geq 0$ or $h \in FS^n, n > 0$. $|t[X/e]| \equiv |h[X/e]| \equiv |h| \equiv h \equiv h[X/|e|] \equiv t[X/|e|]$

Inductive step

– $t \equiv c\ t_1 \ldots t_m, c \in CS^n, 0 \leq m \leq n$. $|t[X/e]| \equiv |(c\ t_1 \ldots t_m)[X/e]|$
$\equiv_{lemma13} |c\ (t_1[X/e]) \ldots (t_m[X/e])| \equiv_{\text{def. of shell}} c\ |t_1[X/e]| \ldots |t_m[X/e]|$
$\equiv_{IH} c\ (t_1[X/|e|]) \ldots (t_m[X/|e|]) \equiv_{lemma13} (c\ t_1 \ldots t_m)[X/|e|] \equiv t[X/|e|]$

– $t \equiv f\ t_1 \ldots t_m, f \in FS^n, 0 \leq m < n$. Very similar to the previous case.

Proof (For Lemma 1, Monotonicity of hypersemantics). By induction on the structure of the context $\mathcal{C}$:

$$Cntxt \ni \mathcal{C} ::= [\,] \mid \mathcal{C}\ e \mid e\ \mathcal{C} \mid let\ X = \mathcal{C}\ in\ e \mid let\ X = e\ in\ \mathcal{C}$$

Base case :

– $\mathcal{C} \equiv []$: then $\mathcal{C}[e] \equiv e$ and $\mathcal{C}[e'] \equiv e'$, therefore the lemma follows from the hypothesis.

Inductive step :

– $\mathcal{C} \equiv \mathcal{C}'\ a$: so $\mathcal{C}[e] \equiv (\mathcal{C}'[e])\ a$. Let $\theta \in PSubst_\perp$ be such that $((\mathcal{C}'[e])\ a)\theta \equiv ((\mathcal{C}'[e])\theta)\ (a\theta) \twoheadrightarrow t$. Then by theorem 2 there must exist $s_1 \in [\![(\mathcal{C}'[e])\theta]\!]$ such that $s_1\ (a\theta) \twoheadrightarrow t$. But by IH $[\![\mathcal{C}'[e]]\!] \Subset [\![\mathcal{C}'[e']]\!]$, so $s_1 \in [\![(\mathcal{C}'[e'])\theta]\!]$ and $((\mathcal{C}'[e'])\theta)\ (a\theta) \twoheadrightarrow t$ by theorem 2 again we have $(\mathcal{C}[e'])\theta \twoheadrightarrow t$.

– $\mathcal{C} \equiv a\ \mathcal{C}'$: so $\mathcal{C}[e] \equiv a\ (\mathcal{C}'[e])$. Let $\theta \in PSubst_\perp$ be such that $(a\ (\mathcal{C}'[e]))\theta \equiv (a\theta)\ ((\mathcal{C}'[e])\theta) \twoheadrightarrow t$. Then by theorem 2 there must exists $s_2 \in [\![(\mathcal{C}'[e])\theta]\!]$ such that $(a\theta)\ s_2 \twoheadrightarrow t$. But by IH $[\![\mathcal{C}'[e]]\!] \Subset [\![\mathcal{C}'[e']]\!]$, so $s_2 \in [\![(\mathcal{C}'[e'])\theta]\!]$ and $(a\theta)\ ((\mathcal{C}'[e'])\theta) \twoheadrightarrow t$ by theorem 2 again we have $(\mathcal{C}[e'])\theta \twoheadrightarrow t$.

– $\mathcal{C} \equiv let\ X = \mathcal{C}'\ in\ e_1$: then $\mathcal{C}[e] \equiv let\ X = \mathcal{C}'[e]\ in\ e_1, \mathcal{C}[e'] \equiv let\ X = \mathcal{C}'[e']\ in\ e_1$. Let $\theta \in PSubst_\perp$ such that $(let\ X = \mathcal{C}'[e]\ in\ e_1)\theta \twoheadrightarrow t$, then the proof must be done using the rule **Let** in the form:

$$\frac{(\mathcal{C}'[e])\theta \twoheadrightarrow t_1\ \ e_1\theta[X/t_1] \twoheadrightarrow t}{let\ X = (\mathcal{C}'[e])\theta\ in\ e_1\theta \twoheadrightarrow t}\ Let$$

As $[\![e]\!] \Subset [\![e']\!]$, then by i.h. (because $\mathcal{C}'$ is a part of $\mathcal{C}$) we have $[\![\mathcal{C}'[e]]\!] \Subset [\![\mathcal{C}'[e']]\!]$ and then $((\mathcal{C}'[e])\theta \twoheadrightarrow t_1)$ implies $((\mathcal{C}'[e'])\theta \twoheadrightarrow t_1)$. Then we have:

$$\frac{\dfrac{}{(\mathcal{C}'[e'])\theta \twoheadrightarrow t_1}\ IH\quad \dfrac{}{e_1\theta[X/t_1] \twoheadrightarrow t}\ Hyp}{let\ X = (\mathcal{C}'[e'])\theta\ in\ e_1\theta \twoheadrightarrow t}\ Let$$

– $\mathcal{C} \equiv let\ X = e_1\ in\ \mathcal{C}'$: then $\mathcal{C}[e] \equiv let\ X = e_1\ in\ \mathcal{C}'[e]$. Let $\theta \in PSubst_\perp$ such that $(let\ X = e_1\ in\ \mathcal{C}'[e])\theta \twoheadrightarrow t$, then the proof must be done by rule **Let** in the form:

$$\frac{e_1\theta \twoheadrightarrow t_1\ \ (\mathcal{C}'[e])\theta[X/t_1] \twoheadrightarrow t}{let\ X = e_1\theta\ in\ (\mathcal{C}'[e])\theta \twoheadrightarrow t}\ Let$$

We have $[\![e]\!] \Subset [\![e']\!]$ and then, by IH, $[\![\mathcal{C}'[e]]\!] \Subset [\![\mathcal{C}'[e']]\!]$. On the other hand $([X/t_1] \circ \theta) \in PSubst_\perp$, so $(\mathcal{C}'[e'])\theta[X/t_1] \twoheadrightarrow t$ and then:

$$\frac{\dfrac{}{e_1\theta \twoheadrightarrow t_1}\ Hyp\quad \dfrac{}{(\mathcal{C}'[e'])\theta[X/t_1] \twoheadrightarrow t}\ IH}{let\ X = e_1\theta\ in\ (\mathcal{C}'[e'])\theta \twoheadrightarrow t}\ Let$$

Proof (For Lemma 2, One-Step Hyper-Soundness of HOlet*-rewriting).* The cases of *(Contx), (Elim), (Bind), (Flat) and (Fapp)* are very similar to the first order case.

(LetAp) $(let\ X = e_1\ in\ e_2)e_3 \rightarrow^l let\ X = e_1\ in\ e_2e_3, \quad$ if $X \notin FV(e_3)$

Given $\theta \in PSusbt_\perp$ *such that* $\mathcal{P} \vdash_{CRWL_{let}} (let\ X = e_1\ in\ e_2e_3)\theta \twoheadrightarrow t$ *then* $\mathcal{P} \vdash_{CRWL_{let}} ((let\ X = e_1\ in\ e_2)e_3)\theta \twoheadrightarrow t$:
If $\mathcal{P} \vdash_{CRWL_{let}} (let\ X = e_1\ in\ e_2e_3)\theta \equiv let\ X = e_1\theta\ in\ (e_2\theta)(e_3\theta) \twoheadrightarrow t$, it must be with a proof:

$$\frac{e_1\theta \twoheadrightarrow t_1 \quad ((e_2\theta)(e_3\theta))[X/t_1] \twoheadrightarrow t}{let\ X = e_1\theta\ in\ (e_2\theta)(e_3\theta) \twoheadrightarrow t}\ Let$$

So we can build a proof for $\mathcal{P} \vdash_{CRWL_{let}} ((let\ X = e_1\ in\ e_2)e_3)\theta \equiv (let\ X = e_1\theta\ in\ e_2\theta)(e_3\theta) \twoheadrightarrow t$:

$$\frac{e_1\theta \twoheadrightarrow t_1 \quad (e_2\theta[X/t_1])(e_3\theta) \equiv ((e_2\theta)(e_3\theta))[X/t_1] \twoheadrightarrow t}{(let\ X = e_1\theta\ in\ e_2\theta)(e_3\theta) \twoheadrightarrow t}\ Let$$

Because, as $X \notin FV(e_3)$ by the premise of (LetAp) and $X \notin vRan(\theta)$ by the variable convention, then $X \notin FV(e_3\theta)$, so $e_3\theta \equiv e_3\theta[X/t_1]$ and also $(e_2\theta[X/t_1])(e_3\theta) \equiv (e_2\theta[X/t_1])(e_3\theta[X/t_1]) \equiv ((e_2\theta)(e_3\theta))[X/t_1]$

(LetIn) $e_1\ e_2 \rightarrow^l let\ X = e_2\ in\ e_1\ X$
if e_2 is an active expression, variable application, junk or $e_2 \equiv let\ Y = e'\ in\ e''$, with $X \in \mathcal{V}$ fresh.

Given $\theta \in PSusbt_\perp$ *such that* $\mathcal{P} \vdash_{CRWL_{let}} (let\ X = e_2\ in\ e_1\ X)\theta \twoheadrightarrow t$ *then* $\mathcal{P} \vdash_{CRWL_{let}} (e_1\ e_2)\theta \twoheadrightarrow t$:
If $\mathcal{P} \vdash_{CRWL_{let}} (let\ X = e_2\ in\ e_1\ X)\theta \twoheadrightarrow t$ it must be with a proof:

$$\frac{e_2\theta \twoheadrightarrow t_2 \quad ((e_1\theta)\ X)[X/t_2] \equiv (e_1\theta)\ t_2 \twoheadrightarrow t}{let\ X = e_2\theta\ in\ (e_1\theta)\ X \twoheadrightarrow t}\ Let$$

By the variable convention and the freshness of X, $(let\ X = e_2\ in\ e_1\ X)\theta \equiv let\ X = e_2\theta\ in\ (e_1\theta)\ X$ and $((e_1\theta)\ X)[X/t_2] \equiv (e_1\theta)\ t_2$. But then by theorem 2, as $t_2 \in [\![e_2\theta]\!]$ and $(e_1\theta)\ t_2 \twoheadrightarrow t$ then it must happen $(e_1e_2)\theta \equiv (e_1\theta)\ (e_2\theta) \twoheadrightarrow t$.

Proof (For Theorem 3, Soundness of HOlet*-rewriting). (i)* Lemma 2, plus an obvious induction over derivation lengths, implies that $[\![e_1]\!] \Subset [\![e_2]\!]$, and then $[\![e_1]\!] \subseteq [\![e_2]\!]$ (just take $\theta = \epsilon$). It can be shown that, for any e', $|e'| \in [\![e']\!]$. But then $|e'| \in [\![e]\!]$, what exactly means that $e \twoheadrightarrow |e'|$.

(ii) From *(i)*, we get $e \twoheadrightarrow |t|$. But $|t| = t$, since t is a pattern.

The following simple lemma will be useful to prove lemma 3:

Lemma 15. *For any* $h \in \Sigma, t_1, \ldots, t_m \in Pat$ *if* $h\ t_1 \ldots t_m \notin Pat$ *then we have* $|h\ t_1 \ldots t_m| = \perp$ *and* $h\ t_1 \ldots t_m$ *is an active expression or junk.*

Proof. By a case distinction:

- $h \in FS^n$: Then $m \geq n$ because otherwise $h\ t_1 \ldots t_m \in Pat$, but then $|h\ t_1 \ldots t_m| = \bot$ because it is an active expression.

- $h \in CS^n$: Then $m > n$ because otherwise $h\ t_1 \ldots t_m \in Pat$, but then $|h\ t_1 \ldots t_m| = \bot$ because it is junk.

Proof (For Lemma 3, Weak peeling lemma). Given $m \geq 0$ let $i : Exp \rightharpoonup \mathbb{N}$ [5] be defined as $i(h\ e_1 \ldots e_m) = \min j \in \{1, \ldots, m\} \mid e_j \in Exp \setminus Pat$ if there exists some $e_j \in Exp \setminus Pat$, $i(h\ e_1 \ldots e_m) = m + 1$ otherwise, and $\pi : Exp \rightharpoonup \mathbb{N}$ be defined as $\pi(h\ e_1 \ldots e_m) = m + 1 - i(h\ e_1 \ldots e_m)$. Then it is easy to prove that for any expression $e \equiv h\ e_1 \ldots e_m$ we have $0 \leq \pi(e) \leq m$. We proceed by induction over the lexicographic product $(size(e), \pi(e))$, where $size(e)$ is equal to the number of symbols of Σ or variables, appering in e.

Base cases $e \equiv h$: Then we are done with $h \rightarrow^{l^0} h$ for $\overline{X} = \emptyset$.

Inductive step $e \equiv h\ e_1 \ldots e_m$ with $m > 0$. If $e_1 \ldots e_m \in Pat$ then we are done with $h\ e_1 \ldots e_m \rightarrow^{l^0} h\ e_1 \ldots e_m$, for $\overline{X} = \emptyset$. Otherwise e has the shape $e \equiv h\ t_1 \ldots t_{i-1}\ e_i \ldots e_m$ with $i > 0$, $t_1, \ldots, t_{i-1} \in Pat, e_i \in Exp \setminus Pat$. Then we can do a case distinction over e_i:

a) $e_i \equiv X\ \overline{e_i}$ with $\overline{e_i} \neq \emptyset$, because otherwise $e_i \equiv X \in Pat$. But then

$$\begin{array}{l} e \equiv ((h\ t_1 \ldots t_{i-1})\ (X\ \overline{e_i}))\ e_{i+1} \ldots e_m \\ \rightarrow^{l} (let\ Y_i = X\ \overline{e_i}\ in\ h\ t_1 \ldots t_{i-1}\ Y_i)\ e_{i+1} \ldots e_m \quad \text{by (LetIn)} \\ \rightarrow^{l^*} let\ Y_i = X\ \overline{e_i}\ in\ h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m \quad \text{by (LetAp*)} \end{array}$$

By (Rule*) we always mean zero o more applications of (Rule). Besides, $size(h\ t_1 \ldots t_{i-1}\ Y_i \ldots e_m) < size(e)$, as $X\ \overline{e_i}$, with $size(X\ \overline{e_i}) \geq 2$ (as $\overline{e_i} \neq \emptyset$), has been replaced by Y_i, with $size(Y_i) = 1$. So, by IH, $h\ t_1 \ldots t_{i-1}\ Y_i \ldots e_m \rightarrow^{l^*} let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m$ under the conditions stipulated, so

$$\begin{array}{l} let\ Y_i = X\ \overline{e_i}\ in\ h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m \\ \rightarrow^{l^*} let\ Y_i = X\ \overline{e_i}\ in\ let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m \end{array}$$

Then $|X\ \overline{e_i}| = \bot$ as $\overline{e_i} \neq \emptyset$, and it is easy the check that all the conditions of the lemma are fulfilled.

[5] We use $\rightharpoonup$ to stress that this is a partial function, as it is only defined for functor applications.

b) $e_i \equiv h_i\ \overline{e_i} \notin Pat, h_i \in \Sigma$. As e properly contains e_i then $size(e_i) < size(e)$, so by IH $e_i \equiv h_i\ \overline{e_i} \to^{l^*} let\ \overline{X_i = a_i}\ in\ h_i\ \overline{t_i}$. But then:

$$
\begin{array}{ll}
e \equiv ((h\ t_1 \ldots t_{i-1})\ (h_i\ \overline{e_i}))\ e_{i+1} \ldots e_m & \\
\to^{l^*} ((h\ t_1 \ldots t_{i-1})\ (let\ \overline{X_i = a_i}\ in\ h_i\ \overline{t_i}))\ e_{i+1} \ldots e_m & \text{(IH)} \\
\to^{l^*} (let\ Y_i = (let\ \overline{X_i = a_i}\ in\ h_i\ \overline{t_i})\ in\ h\ t_1 \ldots t_{i-1}\ Y_i)\ e_{i+1} \ldots e_m & \text{(LetIn)} \\
\to^{l^*} let\ Y_i = (let\ \overline{X_i = a_i}\ in\ h_i\ \overline{t_i})\ in\ h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m & \text{(LetAp*)} \\
\to^{l^*} let\ \overline{X_i = a_i}\ in\ let\ Y_i = h_i\ \overline{t_i}\ in\ h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m & \text{(Flat*)}
\end{array}
$$

Besides, $size(h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m) \leq size(e)$, as e_i, with $size(e_i) \geq 1$, has been replaced by Y_i, with $size(Y_i) = 1$. But as $t_1, \ldots, t_{i-1}, Y_i \in Pat$ then $\pi(h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m) \leq m+1-(i+1) = m-i < m+1-i = \pi(h\ t_1 \ldots t_{i-1}\ e_i \ldots e_m) = \pi(e)$, so by IH $h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m \to^{l^*} let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m$ under the conditions stipulated, so

$$
\begin{array}{l}
let\ \overline{X_i = a_i}\ in\ let\ Y_i = h_i\ \overline{t_i}\ in\ h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m \\
\to^{l^*} let\ \overline{X_i = a_i}\ in\ let\ Y_i = h_i\ \overline{t_i}\ in\ let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m \quad \text{by IH}
\end{array}
$$

And then we have two possibilities:

i) $h_i\ \overline{t_i} \in Pat$: Then

$$
\begin{array}{l}
let\ \overline{X_i = a_i}\ in\ let\ Y_i = h_i\ \overline{t_i}\ in\ let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m \\
\to^{l} let\ \overline{X_i = a_i}\ in\ (let\ \overline{X = a}\ in\ h\ t_1 \ldots t_m)[Y_i / h_i\ \overline{t_i}] \qquad \text{by (Bind)}
\end{array}
$$

Note that by $t_i \equiv Y_i$ because $Y_i \in Pat$, as Y_i was fresh then it does not appear in $t_1, \ldots, t_{i-1}, e_{i-1}, \ldots, e_m$, so as $\overline{a}$ and $t_{i+1}, \ldots, t_m$ come from the IH applied to $h\ t_1 \ldots t_{i-1}\ Y_i\ e_{i+1} \ldots e_m$ then Y_i cannot appear in $\overline{a}$ nor $t_{i+1}, \ldots, t_m$. So

$$
\begin{array}{l}
let\ \overline{X_i = a_i}\ in\ (let\ \overline{X = a}\ in\ h\ t_1 \ldots t_{i-1}\ Y_i\ t_{i+1} \ldots t_m)[Y_i / h_i\ \overline{t_i}] \\
\equiv let\ \overline{X_i = a_i}\ in\ let\ \overline{X = a}\ in\ h\ t_1 \ldots t_{i-1}\ (h_i\ \overline{t_i})\ t_{i+1} \ldots t_m
\end{array}
$$

and it is easy the check that all the conditions of the lemma are fulfilled.

ii) $h_i\ \overline{t_i} \notin Pat$: Then by lemma 15 we have $|h_i\ \overline{t_i}| = \bot$, so it is easy the check that all the conditions of the lemma are fulfilled.

Proof (For Lemma 4). The consequence $t \sqsubseteq t'[\overline{X/\ \bot}]$ holds just applying lemma 14. For the rest of the lemma we proceed by induction on the size K of the proof for $\mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t$, measured as the number of rules applied.

Base cases The reasoning is identical to the FO case.

Inductive step Let us see which rule was applied in the root of the proof for $\mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t$:

DC Then we have $e \equiv h\ e_1 \ldots e_m$ with $m > 0$, $h \in CS^n$ with $m \leq n$ or $h \in FS^n$ with $m < n$, and the following proof:

$$\frac{e_1 \twoheadrightarrow t_1 \quad \ldots \quad e_m \twoheadrightarrow t_m}{\underbrace{h\ e_1 \ldots e_m}_{e} \twoheadrightarrow \underbrace{h\ t_1 \ldots t_m}_{t}}\ DC$$

But then appliing Th. 1 over $h\ e_1 \ldots e_m \twoheadrightarrow h\ t_1 \ldots t_m$ we get:

- $h\, e_1 \ldots e_{m-1} \twoheadrightarrow s_1$ for some $s_1 \in Pat_\perp$ and with $size(h\, e_1 \ldots e_{m-1} \twoheadrightarrow s_1) < K$.
- $s_1\ e_m \twoheadrightarrow h\ t_1 \ldots t_m$ with $size(s_1\ e_m \twoheadrightarrow h\ t_1 \ldots t_m) \leq K$.

As $s_1\ e_m \twoheadrightarrow h\ t_1 \ldots t_m \not\equiv \perp$ then $s_1 \not\equiv \perp$. But then, as by the conditions of **DC** it must happen $h \in CS^n$ with $m \leq n$ or $h \in FS^n$ with $m < n$, the only rule that could have been applied at the root of $h\, e_1 \ldots e_{m-1} \twoheadrightarrow s_1$ is **DC**, so $s_1 \equiv h\, u_1 \ldots u_{m-1}$ for some $u_1, \ldots, u_{m-1} \in Pat_\perp$. But then, with a similar reasoning, as $h\ t_1 \ldots t_m \not\equiv \perp$ then $s_1\ e_m \equiv h\ u_1 \ldots u_{m-1}\ e_m \twoheadrightarrow h\ t_1 \ldots t_m$ must be proved applying **DC** at the root, with a proof like the following:

$$\frac{u_1 \twoheadrightarrow t_1 \quad \ldots \quad u_{m-1} \twoheadrightarrow t_{m-1} \quad e_m \twoheadrightarrow t_m}{\underbrace{h\ u_1 \ldots u_{m-1}}_{s_1}\ e_m \twoheadrightarrow \underbrace{h\ t_1 \ldots t_m}_{t}}\ DC$$

Besides, as each $u_i \in Pat_\perp$ and $u_i \twoheadrightarrow t_i$ then $t_i \sqsubseteq u_i$ for $i \in \{1, \ldots, m-1\}$. Now, as $size(h\ e_1 \ldots e_{m-1} \twoheadrightarrow s_1) < K$ and $s_1 \not\equiv \perp$ then by IH $h\ e_1 \ldots e_{m-1} \rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in\ s'_1$ under the conditions stipulated, with $s_1 \equiv h\ u_1 \ldots u_{m-1} \sqsubseteq s'_1[\overline{X_1/\perp}]$, so $s'_1 \equiv h\ u'_1 \ldots u'_{m-1}$ with $u_i \sqsubseteq u'_i[\overline{X_1/\perp}]$ for $i \in \{1, \ldots, m-1\}$. So we can do:

$$\begin{array}{ll} & h\ e_1 \ldots e_m \\ \rightarrow^l & (let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1})\ e_m \quad \text{by IH} \\ \rightarrow^l & let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ e_m \quad \text{by (LetAp*)} \end{array}$$

Now we have to do a case distinction over t_m:

a) If $t_m \not\equiv \perp$ then as $size(e_m \twoheadrightarrow t_m) < size(s_1\ e_m \twoheadrightarrow h\ t_1 \ldots t_m) \leq K$ we can apply the IH getting $e_m \rightarrow^{l^*} let\ \overline{X_m = a_m}\ in\ t'_m$ under the conditions stipulated. We have two possibilities:

   - If $\overline{X_m} = \emptyset$ then

     $let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ e_m \rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ t'_m$

     and we are done, as $h\ u'_1 \ldots u'_{m-1}\ t'_m \in Pat$, $\overline{a_1} \subseteq Exp$ and $|\overline{a_1}| = \overline{\perp}$ by IH, and $t \equiv h\ t_1 \ldots t_m \sqsubseteq (h\ u'_1 \ldots u'_{m-1}\ t'_m)[\overline{X_1/\perp}]$

because $t_i \sqsubseteq u_i \sqsubseteq u'_i[\overline{X_1}/\ \bot]$ by IH, $t_m \sqsubseteq t'_m$ by IH, and $t'_m \equiv t'_m[\overline{X_1}/\ \bot]$ as $\overline{X_1}$ are were created fresh in a derivation in a position parallel with respect to the position of e_m.

– If $\overline{X_m} \neq \emptyset$ then we can do:

$$
\begin{array}{ll}
& let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ e_m \\
\rightarrow^{l^*} & let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}(let\ \overline{X_m = a_m}\ in\ t'_m) \quad \text{by IH} \\
\rightarrow^{l} & let\ \overline{X_1 = a_1}\ in \\
& \quad let\ Y = (let\ \overline{X_m = a_m}\ in\ t'_m)\ in\ h\ u'_1 \ldots u'_{m-1}\ Y \quad \text{by (LetIn)} \\
\rightarrow^{l^*} & let\ \overline{X_1 = a_1}\ in \\
& \quad let\ \overline{X_m = a_m}\ in\ let\ Y = t'_m\ in\ h\ u'_1 \ldots u'_{m-1}\ Y \quad \text{by (Flat*)} \\
\rightarrow^{l} & let\ \overline{X_1 = a_1}\ in\ let\ \overline{X_m = a_m}\ in\ h\ u'_1 \ldots u'_{m-1}\ t'_m \quad \text{by (Bind)}
\end{array}
$$

And we are done, it is very easy to see that all the conditions are fulfilled reasoning in a similar way as we did in the previous case.

b) If $t_m \equiv \bot$ we do a case distinction over e_m:

i) $e_m \in Pat$. Then we are done as $h\ u'_1 \ldots u'_{m-1} \in Pat$ by IH, which implies $h\ u'_1 \ldots u'_{m-1}\ e_m \in Pat$; $\overline{a_1} \subseteq Exp$ and $|\overline{a_1}| = \overline{\bot}$ by IH, and $t \equiv h\ t_1 \ldots t_m \sqsubseteq (h\ u'_1 \ldots u'_{m-1}\ e_m)[\overline{X_1}/\ \bot]$ because $t_i \sqsubseteq u_i \sqsubseteq u'_i[\overline{X_1}/\ \bot]$ by IH, $t_m \equiv \bot \sqsubseteq e_m$, and $e_m \equiv e_m[\overline{X_1}/\ \bot]$ as $\overline{X_1}$ are were created fresh in a derivation in a position parallel with respect to the position of e_m.

ii) $e_m \equiv X_m\ \overline{e_m}$. Then we can do:

$$
\begin{array}{ll}
& let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ (X_m\ \overline{e_m}) \\
\rightarrow^{l^*} & let\ \overline{X_1 = a_1}\ in\ let\ Z = X_m\ \overline{e_m}\ in\ h\ u'_1 \ldots u'_{m-1}\ Z \quad \text{by (LetIn)}
\end{array}
$$

And we are done, it is very easy to see that all the conditions are fulfilled reasoning in a similar way as we did in the previous case, taking account of $|X_m\ \overline{e_m}| = \bot$ and $\bot \sqsubseteq X_m\ \overline{e_m}$.

iii) $e_m \equiv h_m\ \overline{e_m} \notin Pat, h_m \in \Sigma$. Then by lemma 3 we have $h_m\ \overline{e_m} \rightarrow^{l^*} let\ \overline{X_m = a_m}\ in\ h_m\ \overline{t'_m}$ such that $\overline{t'_m} \subseteq Pat$, $\overline{a_m} \subseteq Exp$ and $|\overline{a_m}| = \overline{\bot}$. Then we have two possibilities:

– If $\overline{X_m} = \emptyset$ then it must happen $\overline{e_m} \subseteq Pat$ because otherwise some *let* should be introduced to fulfill lemma 3. So by lemma 15 we have $|h_m\ \overline{e_m}| = \bot$ being $h_m\ \overline{e_m}$ active or junk, and we can do:

$$
\begin{array}{ll}
& let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ (h_m\ \overline{e_m}) \\
\rightarrow^{l} & let\ \overline{X_1 = a_1}\ in\ let\ Z = h_m\ \overline{e_m}\ in\ h\ u'_1 \ldots u'_{m-1}\ Z \quad \text{by (LetIn)}
\end{array}
$$

And we are done, it is very easy to see that all the conditions are fulfilled reasoning in a similar way as we did in the previous case, taking account of $|h_m\ \overline{e_m}| = \bot$ and $\bot \sqsubseteq h_m\ \overline{e_m}$.

– If $\overline{X_m} \neq \emptyset$ then we can do:

$$
\begin{array}{l}
let\ \overline{X_1 = a_1}\ in\ h\ u'_1 \ldots u'_{m-1}\ (h_m\ \overline{e_m}) \\
\rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in \\
\quad h\ u'_1 \ldots u'_{m-1}\ (let\ \overline{X_m = a_m}\ in\ h_m\ \overline{t'_m}) \quad \text{by lemma 3} \\
\rightarrow^{l} let\ \overline{X_1 = a_1}\ in \\
\quad let\ Z = (let\ \overline{X_m = a_m}\ in\ h_m\ \overline{t'_m})\ in \\
\quad\quad h\ u'_1 \ldots u'_{m-1}\ Z \qquad\qquad \text{by (LetIn)} \\
\rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in \\
\quad let\ \overline{X_m = a_m}\ in \\
\quad\quad let\ Z = h_m\ \overline{t'_m}\ in\ h\ u'_1 \ldots u'_{m-1}\ Z \quad \text{by (Flat*)}
\end{array}
$$

If $h_m\ \overline{t'_m} \notin Pat$ we are done, because as $h_m \in \Sigma$ and $\overline{t'_m} \subseteq Pat$ then $|h_m\ \overline{t'_m}| = \bot$ by lemma 15, and it is very easy to see that all the conditions are fulfilled reasoning in a similar way as we did in the previous case, taking account of $\overline{a_m} \subseteq Exp$ and $|\overline{a_m}| = \overline{\bot}$ by lemma 3, and $\bot \sqsubseteq Z$.

On the other hand, if $h_m\ \overline{t'_m} \in Pat$ we can now do

$$
\begin{array}{l}
let\ \overline{X_1 = a_1}\ in \\
\quad let\ \overline{X_m = a_m}\ in\ let\ Z = h_m\ \overline{t'_m}\ in\ h\ u'_1 \ldots u'_{m-1}\ Z \\
\rightarrow^{l} let\ \overline{X_1 = a_1}\ in \\
\quad let\ \overline{X_m = a_m}\ in\ h\ u'_1 \ldots u'_{m-1}\ (h_m\ \overline{t'_m}) \quad \text{by (Bind)}
\end{array}
$$

And we are done, it is very easy to see that all the conditions are fulfilled reasoning in a similar way as we did in the previous case, taking account of $\bot \sqsubseteq h_m\ \overline{t'_m}$.

OR Then we have case $e \equiv f\ e_1 \ldots e_n\ g_1 \ldots g_m$. In case $n = 0$ we have the following proof:

$$\frac{(r\sigma)g_1 \ldots g_m \twoheadrightarrow t}{f\ g_1 \ldots g_m \twoheadrightarrow t}\ OR$$

for $(f \rightarrow r)\sigma \in [\mathcal{P}]_\bot$. But then we can define σ' as the substitution built from σ replacing every $\bot$ that appears in some expresion in $ran(\sigma)$ with some fresh variable. Then $\sigma \sqsubseteq \sigma'$ and $\sigma' \in PSubst$, so by monotonicity of HOCRWL-derivability, we have $(r\sigma')g_1 \ldots g_m \twoheadrightarrow t$ with a proof of the same size of $(r\sigma)g_1 \ldots g_m \twoheadrightarrow t$, so we can apply the IH getting $(r\sigma')g_1 \ldots g_m \rightarrow^{l^*} let\ \overline{X = a}\ in\ t'$ under the conditions stipulated. But then:

$$
\begin{array}{l}
f\ g_1 \ldots g_m \\
\rightarrow^{l} (r\sigma')\ g_1 \ldots g_m \quad \text{by (Fapp)} \\
\rightarrow^{l} let\ \overline{X = a}\ in\ t' \quad \text{by IH}
\end{array}
$$

and we are done.

In case $n > 0$ we can apply lema 1 to $f\ e_1 \ldots e_n\ g_1 \ldots g_m \twoheadrightarrow t$ getting:

- $f\ e_1 \ldots e_{n-1} \twoheadrightarrow s_1$ for some $s_1 \in Pat_\bot$ and with $size(f\ e_1 \ldots e_{n-1} \twoheadrightarrow s_1) < K$.
- $s_1\ e_n\ g_1 \ldots g_m \twoheadrightarrow t$ with $size(s_1\ e_n\ g_1 \ldots g_m \twoheadrightarrow t) \leq K$.

As $s_1\ e_n\ g_1 \ldots g_m \twoheadrightarrow t \not\equiv \bot$ then $s_1 \not\equiv \bot$. But then, as $f \in FS^n$ (because we require that the program rules respect the arity of the functions) and $s_1 \not\equiv \bot$, the only rule that could have been applied at the root of $f\ e_1 \ldots e_{n-1} \twoheadrightarrow s_1$ is **DC**, so $s_1 \equiv f\ u_1 \ldots u_{n-1}$ for some $u_1, \ldots, u_{n-1} \in Pat_\bot$. But then, with a similar reasoning, as $s_1\ e_n\ g_1 \ldots g_m \equiv f\ u_1 \ldots u_{n-1}\ e_n\ g_1 \ldots g_m$ is a total or over application then $s_1\ e_n\ g_1 \ldots g_m \twoheadrightarrow t$ must be proved applying **OR** at the root, with a proof like the following:

$$\frac{u_1 \twoheadrightarrow t_1 \quad \ldots \quad u_{n-1} \twoheadrightarrow t_{n-1} \quad e_n \twoheadrightarrow t_n \quad (r\sigma)\ g_1 \ldots g_m \twoheadrightarrow t}{\underbrace{f\ u_1 \ldots u_{n-1}}_{s_1}\ e_n\ g_1 \ldots g_m \twoheadrightarrow t}\ OR$$

for $(f\ p_1 \ldots p_n \to r)\sigma \in [\mathcal{P}]_\bot$ with $\overline{p}\sigma \equiv \overline{t}$ and $\overline{p} \subseteq Pat$ lineal. Besides, as each $u_i \in Pat_\bot$ and $u_i \twoheadrightarrow t_i$ then $t_i \sqsubseteq u_i$ for $i \in \{1, \ldots, n-1\}$. Now, as $size(f\ e_1 \ldots e_{n-1} \twoheadrightarrow s_1) < K$ and $s_1 \not\equiv \bot$ then by IH $f\ e_1 \ldots e_{n-1} \to^{l*} let\ \overline{X_1 = a_1}\ in\ s_1'$ under the conditions stipulated, with $s_1 \equiv f\ u_1 \ldots u_{n-1} \sqsubseteq s_1'[\overline{X_1/\ \bot}]$, so $s_1' \equiv f\ u_1' \ldots u_{n-1}'$ with $u_i \sqsubseteq u_i'[\overline{X_1/\ \bot}]$ for $i \in \{1, \ldots, m-1\}$. So we can do:

$$\begin{array}{ll} & f\ e_1 \ldots e_n\ g_1 \ldots g_m \\ \to^l & (let\ \overline{X_1 = a_1}\ in\ f\ u_1' \ldots u_{n-1}')\ e_n\ g_1 \ldots g_m \quad \text{by IH} \\ \to^l & let\ \overline{X_1 = a_1}\ in\ f\ u_1' \ldots u_{n-1}'\ e_n\ g_1 \ldots g_m \quad \text{by (LetAp*)} \end{array}$$

Now we have to do a case distinction over t_n:

a) If $t_n \not\equiv \bot$ then as $size(e_n \twoheadrightarrow t_n) < size(s_1\ e_n\ g_1 \ldots g_m \twoheadrightarrow t) \leq K$ we can apply the IH getting $e_n \to^{l*} let\ \overline{X_n = a_n}\ in\ t_n'$ with $t_n \sqsubseteq t_n'[\overline{X_n/\ \bot}]$, under the conditions stipulated. But then

$$\begin{array}{l} let\ \overline{X_1 = a_1}\ in\ f\ u_1' \ldots u_{n-1}'\ e_n\ g_1 \ldots g_m \\ \to^{l*} let\ \overline{X_1 = a_1}\ in\ let\ \overline{X_n = a_n}\ in\ f\ u_1' \ldots u_{n-1}'\ t_n'\ g_1 \ldots g_m \end{array}$$

in a similar way as we did in the case for **DC**, using (LetIn), (Flat*) and (Bind) in case $\overline{X_n} \neq \emptyset$. Besides, as $\overline{t} \sqsubseteq u_1, \ldots, u_{n-1}, t_n'[\overline{X_n/\ \bot}] \sqsubseteq u_1'[\overline{X_1/\ \bot}], \ldots, u_{n-1}'[\overline{X_1/\ \bot}], t_n'[\overline{X_n/\ \bot}] \sqsubseteq u_1', \ldots, u_{n-1}', t_n'$ there exists some $\sigma' \in PSubst$ such that $\overline{p}\sigma' \equiv u_1', \ldots, u_{n-1}', t_n'$ and $\sigma \sqsubseteq \sigma'$. But then, as $(r\sigma)\ g_1 \ldots g_m \twoheadrightarrow t$ then $(r\sigma')\ g_1 \ldots g_m \twoheadrightarrow t$ with a proof of the same size. As $size((r\sigma)\ g_1 \ldots g_m \twoheadrightarrow t) < size(s_1\ e_n\ g_1 \ldots g_m \twoheadrightarrow t) \leq K$ we can apply the IH to $(r\sigma')\ g_1 \ldots g_m \twoheadrightarrow t$ getting $(r\sigma')\ g_1 \ldots g_m \to^{l*} let\ \overline{X = a}\ in\ t'$

under the conditions stipulated, so we can chain:

$$\begin{array}{ll} let\ \overline{X_1 = a_1}\ in\ let\ \overline{X_n = a_n}\ in\ f\ u'_1 \dots u'_{n-1}\ t'_n\ g_1 \dots g_m & \\ \rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in\ let\ \overline{X_n = a_n}\ in\ (r\sigma')\ g_1 \dots g_m \dots g_m & \text{by (Fapp)} \\ \rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in\ let\ \overline{X_n = a_n}\ in\ let\ \overline{X = a}\ in\ t' & \text{by IH} \end{array}$$

And we are done, it is very easy to see that all the conditions are fulfilled.

b) If $t_n \equiv \bot$ we do a case distinction over e_n:

i) $e_n \in Pat$. Then $t_1, \dots, t_{n-1}, \bot \sqsubseteq u'_1, \dots, u'_{n-1}, e_n$ and $u'_1, \dots, u'_{n-1}, e_n \subseteq Pat$ and then there exists some $\sigma' \in PSubst$ to apply (Fapp)with which proceed as we did in the previous case.

ii) $e_n \equiv X_n\ \overline{e_n}$. Then we can do:

$$\begin{array}{l} let\ \overline{X_1 = a_1}\ in\ f\ u'_1 \dots u'_{n-1}\ (X_n\ \overline{e_n})\ g_1 \dots g_m \\ \rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in\ let\ Z = X_n\ \overline{e_n}\ in\ f\ u'_1 \dots u'_{n-1}\ Z\ g_1 \dots g_m \end{array}$$

Using (LetIn), (LetAp*). But then $t_1, \dots, t_{n-1}, \bot \sqsubseteq u'_1, \dots, u'_{n-1}, Z$ and $u'_1, \dots, u'_{n-1}, Z \subseteq Pat$, so we can proceed like in the previous case.

iii) $e_n \equiv h_n\ \overline{e_n} \notin Pat, h_n \in \Sigma$. Then we can proceed in a similar way as we did in **DC**, applying lemma 3 to get

$$\begin{array}{l} let\ \overline{X_1 = a_1}\ in\ f\ u'_1 \dots u'_{n-1}\ (h_n\ \overline{e_n})\ g_1 \dots g_m \\ \rightarrow^{l^*} let\ \overline{X_1 = a_1}\ in\ let\ \overline{X_n = a_n}\ in\ f\ u'_1 \dots u'_{n-1}\ t'_n\ g_1 \dots g_m \end{array}$$

For some $t'_n \in Pat$, so we can proceed like in cases *i)* and *ii)* applying (Fapp) and the HI over $(r\sigma')\ g_1 \dots g_m \twoheadrightarrow t$.

Proof (For Theorem 4, Completeness of HOlet*-rewriting).* Part *(i)* is simply Lemma 4, taking $e' \equiv let\ \overline{X = a}\ in\ t'$. For part *(ii)*, Lemma 4 ensures $t \sqsubseteq t'[\overline{X/\ \bot}]$. But since t is total, it must be the case that $t \equiv t'[\overline{X/\ \bot}]$, and therefore the variables $\overline{X}$ cannot appear in t', and we conclude that $t \equiv t'$. But then we can apply *(Elim)* to $e' \equiv let\ \overline{X = a}\ in\ t$, obtaining $e' \rightarrow^{l} t$, and therefore $e \rightarrow^{l^*} t$.

Proof (of Theorem 5, Equivalence of HOlet*-rewriting and* HOCRWL*-derivability).* This result simply joins together part *(ii)* of Theorems 3 and 4.

4 Higher order *let*-narrowing

Proof (For Lemma 5, Closedness of $\rightarrow^{L}$ under PSubst). The proof is almost identical to proof for the first order version of $\rightarrow^{l}$, just adding a straightforward case for (LetAp), and using the fact that for (LetIn') it does not matter if e_1 was a variable application and $e_1\theta \in Pat$, because we can extract it to a *let* anyway.

Proof (For Theorem 6, Soundness of $\rightsquigarrow^l$ wrt $\rightarrow^L$). It is easier to prove the result for a variant of $\rightsquigarrow^l$ that uses (VNarr) (see Sect. 4) instead of (VAct) and (VBind). Since that variant defines a larger relation, proving soundness for it implies proving soundness also for the original (smaller) relation. The detailed definition for (VNarr) is:

(VNarr) $X\ e \rightsquigarrow^L_{[X/t]}\ t\ (e[X/t])$, for any $t \in Pat$

besides, when combined with (Contx) it must happen that $var(t)$ are fresh and $X \notin BV(\mathcal{C})$.

First we will prove the soundness of narrowing for one step, and see that for one step the lemma is true using $\rightarrow^l$ instead of $\rightarrow^L$, that is, that $e \rightsquigarrow^L_\theta e'$ implies $e\theta \rightarrow^l e'$, if the rule (VNarr) was not used, or $e\theta \rightarrow^{l^0} e'$ if the rule (VNarr) was used. We proceed by a case distinction over the rule used in $e \rightsquigarrow^L_\theta e'$.

(X) Very similar as the first order case, new cases for the rules (LetAp) and (LetIn) are straightforward.

(Narr) Then we have $f\ \bar{t} \rightsquigarrow^L_\theta r\theta$ for $(f\ \overline{p} \rightarrow r) \in \mathcal{P}$ fresh, $\theta \in PSubst$ such that $\bar{t}\theta \equiv \overline{p}\theta$. But then $(f\ \overline{p} \rightarrow r)\theta \in [\mathcal{P}]$ and besides $(f\ \overline{p} \rightarrow r)\theta \equiv f\ \overline{p}\theta \rightarrow r\theta \equiv f\ \bar{t}\theta \rightarrow r\theta$, so $(f\ \bar{t} \rightarrow r)\theta \in [\mathcal{P}]$ and we can do $e\theta \equiv f\ \bar{t}\theta \rightarrow^l r\theta \equiv e'$, by (Fapp).

(VNarr) Then we have $X\ a \rightsquigarrow^L_{[X/t]} t\ (a[X/t])$. But then $e\theta \equiv (X\ a)[X/t] \equiv t\ (a[X/t]) \equiv e'$, so $e\theta \rightarrow^{l^0} e\theta \equiv e'$.

(Contxt) Then we have $\mathcal{C}[e] \rightsquigarrow^L_\theta \mathcal{C}\theta[e']$ because $e \rightsquigarrow^L_\theta e'$. Let us do a case distintion over the rule applied in $e \rightsquigarrow^L_\theta e'$:

a) $e \rightsquigarrow^L_\theta e' \equiv f\ \bar{t} \rightsquigarrow^L_\theta r\theta$ by (Narr), for $(f\ \overline{p} \rightarrow r) \in \mathcal{P}$ fresh, so $f\ \bar{t}\theta \rightarrow^l r\theta$ by (Fapp), as we saw in the case for (Narr). Then $(\mathcal{C}[e])\theta \equiv (\mathcal{C}[e])\theta|_{\setminus var(\overline{p})}$, because the variables in $var(\overline{p})$ are fresh as $(f\ \overline{p} \rightarrow r)$ is. But then, as $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ and $vRan(\theta|_{\setminus var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$ by the conditions in (Contx), and $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ implies $dom(\theta|_{\setminus var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$, we can apply lemma 11 getting $(\mathcal{C}[e])\theta|_{\setminus var(\overline{p})} \equiv \mathcal{C}\theta|_{\setminus var(\overline{p})}[e\theta|_{\setminus var(\overline{p})}] \equiv \mathcal{C}\theta|_{\setminus var(\overline{p})}[f\ \bar{t}\theta|_{\setminus var(\overline{p})}] \equiv \mathcal{C}\theta[f\ \bar{t}\theta]$, because the variables in $var(\overline{p})$ are fresh again. Besides $vRan(\theta|_{\setminus var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$, so we can appply (Contx) combined with an inner (Fapp) step to do $(\mathcal{C}[e])\theta \equiv \mathcal{C}\theta[f\ \bar{t}\theta] \rightarrow^l \mathcal{C}\theta[r\theta] \equiv \mathcal{C}\theta[e']$.

b) $e \rightsquigarrow^L_\theta e' \equiv X\ a \rightsquigarrow^L_{[X/t]} t\ (a[X/t])$ by (VNarr). Then, as $X \notin BV(\mathcal{C})$ and $var(t)$ fresh by the conditions in (Contx), then $dom([X/t]) \cap BV(\mathcal{C}) = vRan([X/t]) \cap BV(\mathcal{C}) = \emptyset$, and so we can apply lemma 11 getting $(\mathcal{C}[e])\theta \equiv (\mathcal{C}[X\ a])[X/t] \equiv \mathcal{C}[X/t][(X\ a)[X/t]] \equiv \mathcal{C}[X/t][t\ (a[X/t])] \equiv \mathcal{C}\theta[e']$.

c) In case a different rule was applied in $e \rightsquigarrow^L_\theta e'$ then $\theta = \epsilon$. Besides, by the proof of the other cases we have $e \equiv e\epsilon \equiv e\theta \rightarrow^l e'$, so $(\mathcal{C}[e])\theta \equiv (\mathcal{C}[e])\epsilon \equiv \mathcal{C}[e] \rightarrow^l \mathcal{C}[e'] \equiv \mathcal{C}\epsilon[e'] \equiv \mathcal{C}\theta[e']$.

Now we will try to prove the lemma for any number of steps $\rightarrow^l$, proceeding by induction over the length n of $e \rightsquigarrow_\theta^{L^n} e'$. The case $e \rightsquigarrow_\epsilon^{L^0} e \equiv e'$ is straightforward because $e \rightarrow^{l^0} e \equiv e'$. The problem comes for $e \rightsquigarrow^L{}_\sigma e'' \rightsquigarrow_\gamma^{L^n} e'$, when trying to chain the induction hypothesis: by IH $e''\gamma \rightarrow^{l^*} e'$, and by the proof for one step $e\sigma \rightarrow^{l^=} e''$, but as $\rightarrow^l$ is not closed under $PSubst$ we cannot chain these steps. But as $\rightarrow^l \subseteq \rightarrow^L$, then we have covered the base case for $\rightarrow^L$, and for the inductive step we have $e''\gamma \rightarrow^{L^*} e'$ and $e\sigma \rightarrow^{l^=} e''$. But then $e\sigma \rightarrow^{l^=} e''$ implies $e\sigma\gamma \rightarrow^{l^=} e''\gamma$, by lemma 5, so we can chain $e\sigma\gamma \rightarrow^{l^=} e''\gamma \rightarrow^{L^*} e'$.

Proof (For theorem 7).

a) If $e \rightsquigarrow_\theta^{L^*} e'$ then $e\theta \rightarrow^{L^*} e'$, by theorem 6, but then $[\![e']\!] \subseteq [\![e\theta]\!]$ by theorem 3.
b) If $e \rightsquigarrow_\theta^{L^*} t$ then $[\![t]\!] \subseteq [\![e\theta]\!]$ by *a)*. But then as $t \in Pat$ implies $t \in [\![t]\!]$ then $t \in [\![e\theta]\!]$, so $e\theta \rightarrow^{l^*} t$ by theorem 4

Proof (For lemma 6). Let us proof the lemma for one narrowing step first, we will see that if $e\theta \rightarrow^l e'$ then $e \rightsquigarrow^l{}_\sigma e'$ under the conditions above. Let us do a case distinction over the rule applied in $e\theta \rightarrow^l e'$:

X $\in \{Elim, Flat, LetIn, LetAp\}$ In this case lifting can be done very easily using $\sigma = \epsilon$ and $\theta' = \theta$, just being a little careful in the case for (LetIn):
- If $e_2\theta$ is a variable application or a *let*-rooted expression then e_2 must be also a variable application or *let*-rooted expression.
- If $e_2\theta$ is active or junk then it might happen:
  a) $e_2 \equiv h\ \overline{t_2}$: Then $h\ \overline{t_2}$ is also junk or active because h in $e_2\theta$ has the same arity and is applied to the same number of arguments.
  b) $e_2 \equiv Y\ \overline{t_2}$: The only problematic case is that in which $\overline{t_2} = \emptyset$ and so $e_2 \equiv Y \in Pat$. But the only way it could happen with $e_2\theta$ being active or junk, is that $\theta = [Y/e']$ for some e' active or junk, but then $\theta \notin PSubst$, which contradicts the hypothesis of the lifting lemma.

Bind Then $e\theta \equiv let\ X = s_1\theta\ in\ s_2\theta \rightarrow^l s_2\theta[X/s_1\theta] \equiv e'$ with $s_1\theta \in Pat$. If $s_1 \in Pat$ also then the proof is straightforward and similar to the previous case. Otherwise we will need the following lemma:

Lemma 16. *Given $e \in LExp \setminus Pat, \theta \in PSubst$ if one has $e\theta \in Pat$ then $e \in Exp$ and $\forall o \in \mathcal{O}(e)$ such that $e|_o \equiv X\ a_1 \ldots a_k$ with $k > 0$ then $\theta(X) \equiv h\ t_1 \ldots t_m$ with $h \in CS^n, m + k \leq n$ or $h \in FS^n, m + k < n$.*

Proof (Sketch). $e \in Exp$ because *let* expression do not disappear after applying substitutions. Using proof by contradiction, if $\theta(X) \equiv h\ t_1 \ldots t_m$ does not hold under those conditions for some X then some subexpression of $e\theta$ is not a pattern.

Let $s_1 \equiv s_1[Z_1\ \overline{a_1}, \ldots, Z_n\ \overline{a_n}]$ be[6]. As $s_1\theta \in Pat$, then by lemma 16 we have $\forall Z_i \in \{Z_1, \ldots, Z_n\}, \theta(Z_i) \equiv h_i\ \overline{t_i}$. Now we define $\sigma(Z_i) \equiv h_i\ \overline{U_i}$ for $\overline{U_i}$ fresh and linear, so:

[6] With $s_1[\ , \ldots, \ ]$ we denote a context with several holes, defined as usual.

- $dom(\sigma) = \overline{Z} \subseteq FV(s_1) = FV(s_1) \setminus \{X\}$, because X cannot appear in s_1 because there are no recursive *lets*.
- $Ran(\sigma)$ contains only fresh shallow patterns. $vRan(\sigma) = \overline{U}$, where $\overline{U}$ is just an alias for $\bigcup_i \overline{U_i}$.
- $s_1\sigma \equiv s_1[h_1\ \overline{U_1}\ \overline{a_1}, \ldots, h_n\ \overline{U_n}\ \overline{a_n}] \in Pat$
- $Z_i\sigma[\overline{U_i}/\overline{t_i}] \equiv h_i\ \overline{U_i}[\overline{U_i}/\overline{t_i}] \equiv h_i\overline{t_i} \equiv Z_i\theta$.

So we can apply (VBind) to do:

$$e \equiv let\ X = s_1\ in\ s_2 \rightsquigarrow^{l}_{\sigma} s_2\sigma[X/s_1\sigma] \equiv e''$$

Let $\theta'_1 = \biguplus_i [\overline{U_i/t_i}] \in PSubst$. Then $dom(\theta'_1) = \overline{U}$ and $\sigma\theta'_1 = \theta[\overline{Z}]$, as we saw before. Now we can define $\theta'_0 = \theta$ and $\theta' = \theta'_0 \uplus \theta'_1$, which is correctly defined as $dom(\theta'_0) = dom(\theta) \subseteq \mathcal{W}$, and $dom(\theta'_1) = \overline{U}$ which are fresh. Now we will see how the conditions in lemma 6 hold:

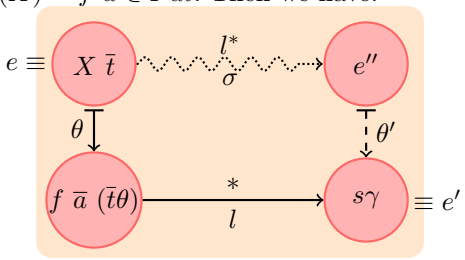
- Condition *ii)* : $\sigma\theta' = \theta[\mathcal{W}]$: Given $Y \in \mathcal{W}$
  - a) If $Y \in \overline{Z}$: Then $Y\theta \equiv Y\sigma\theta'_1$ because $\sigma\theta'_1 = \theta[\overline{Z}]$. As $Y \in \overline{Z}$ then $Y \in dom(\sigma)$ and $var(Y\sigma)$ are fresh and do not appear in $dom(\theta'_0) = dom(\theta)$. But then $Y\sigma\theta'_1 = Y\sigma\theta$.
  - b) If $Y \in \mathcal{W} \setminus \overline{Z}$: Then $Y \notin dom(\sigma)$ and $var(Y\sigma) \cap dom(\theta'_1) = \emptyset$ by definition, so by definition of θ'_0 we have $Y\theta \equiv Y\theta'_0 \equiv Y\theta' \equiv Y\sigma\theta'$.
- Condition *i)* : $e''\theta' \equiv e'$: By the variable convention $X \notin (dom(\theta) \cup vRan(\theta))$, so we can apply lemma 10 over $e' \equiv s_2\theta[X/s_1\theta] \equiv s_2[X/s_1]\theta$. We have seen $X \notin dom(\sigma)$ and besides $X \notin vRan(\sigma)$, so we also can apply lemma 10 to get $e'' \equiv s_2\sigma[X/s_1\sigma] \equiv s_2[X/s_1]\sigma$. But $\mathcal{W} \supseteq FV(e) = FV(let\ X = s_1\ in\ s_2) = FV(s_1) \cup (FV(s_2) \setminus \{X\}) \supseteq FV(s_2[X/s_1])$. So, as $\sigma\theta' = \theta[\mathcal{W}]$, we have:

$$\underbrace{s_2[X/s_1]\sigma}_{e''}\,\theta' \equiv \underbrace{s_2[X/s_1]\theta}_{e'}$$

- Condition *iii)* : $(dom(\theta') \cup vRan(\theta')) \cap \mathcal{B} = \emptyset$: Remember $\theta' = \theta'_0 \uplus \theta'_1$. Regarding θ'_0 the condition holds as $\theta'_0 = \theta$, and $(dom(\theta) \cup vRan(\theta)) \cap \mathcal{B} = \emptyset$ by the hypothesis. Regading θ'_1, if $Y \in dom(\theta'_1) = \overline{U}$ then Y is fresh and so $Y \notin \mathcal{B}$. On the other hand, $vRan(\theta'_1) = var(\bigcup_i \overline{t_i}) \subseteq vRan(\theta)$, and $vRan(\theta) \cap \mathcal{B} = \emptyset$ by the hypothesis, so $vRan(\theta'_1) \cap \mathcal{B} = \emptyset$.

Fapp Then e can have two possible shapes:

1. $e \equiv X\ \overline{t}$, with $\theta(X) = f\ \overline{a} \in Pat$. Then we have:



With an (Fapp) step $e\theta \equiv f\ \overline{a}\ (\overline{t}\theta) \rightarrow^l s\gamma$ using a fresh variant $(f\ \overline{p} \rightarrow s) \in \mathcal{P}$ such that $(X\ \overline{t})\theta \equiv (f\ \overline{p})\gamma$ for some $\gamma \in PSubst$. We can assume that $dom(\gamma) \subseteq FV(f\ \overline{p} \rightarrow s)$ without loss of generality. But then $dom(\theta) \cap dom(\gamma) = \emptyset$, and so $\theta \uplus \gamma$ is correctly defined, and it is a unifier of $X\ \overline{t}$ and $f\ \overline{p}$. So, there must exists $\sigma = mgu(X\ \overline{t}, f\ \overline{p})$, which we can use to perform a (VAct) step, because $\theta \in PSubst$ and $(X\ \overline{t})\theta$ active imply $\overline{t} \neq \emptyset$:

$$e \equiv X\ \overline{t} \rightsquigarrow^l_\sigma\ s\sigma \equiv e''$$

As this unifier is an mgu then $dom(\sigma) \subseteq FV(X\ \overline{t}) \cup FV(f\ \overline{p})$, $vRan(\sigma) \subseteq FV(X\ \overline{t}) \cup FV(f\ \overline{p})$ and $\sigma \lesssim (\theta \uplus \gamma)$, so there must exists $\theta'_1 \in PSubst$ such that $\sigma\theta'_1 = \theta \uplus \gamma$. Besides we can define $\theta'_0 = \theta|_{\setminus (dom(\theta'_1) \cup FV(X\ \overline{t}))}$ and then we can take $\theta' = \theta'_0 \uplus \theta'_1$ which is correctly defined as obviously $dom(\theta'_0) \cap dom(\theta'_1) = \emptyset$. Besides $dom(\theta'_0) \cap (FV(X\ \overline{t}) \cup FV(f\ \overline{p})) = \emptyset$, as if $Y \in FV(X\ \overline{t})$ then $Y \notin dom(\theta'_0)$ by definition; and if $Y \in FV(f\ \overline{p})$ then $Y \notin dom(\theta)$ as $\overline{p}$ belong to the fresh variant, and so $Y \notin dom(\theta'_0)$. Then the conditions in lemma 6 hold:

- Condition *i)* : $e''\theta' \equiv e'$: As $e''\theta' \equiv s\sigma\theta' \equiv s\sigma\theta'_1$ because given $Y \in FV(s\sigma)$, if $Y \in FV(s)$ then it belongs to the fresh variant and so $Y \notin dom(\theta) \supseteq dom(\theta'_0)$; and if $Y \in vRan(\sigma) \subseteq FV(X\ \overline{t}) \cup FV(f\ \overline{p})$ then $Y \notin dom(\theta'_0)$ because $dom(\theta'_0) \cap (FV(X\ \overline{t}) \cup FV(f\ \overline{p})) = \emptyset$. But $s\sigma\theta'_1 \equiv s(\theta \uplus \gamma) \equiv s\gamma \equiv e'$, because $\sigma\theta'_1 = \theta \uplus \gamma$ and s is part of the fresh variant.
- Condition *ii)* : $\sigma\theta' = \theta[\mathcal{W}]$: Given $Y \in \mathcal{W}$, if $Y \in FV(X\ \overline{t})$ then $Y \notin dom(\gamma)$ and so $Y\theta \equiv Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1$, as $\sigma\theta'_1 = \theta \uplus \gamma$. But $Y\sigma\theta'_1 \equiv Y\sigma\theta'$ because given $Z \in var(Y\sigma)$, if $Z \equiv Y$ then as $Y \in FV(X\ \overline{t})$ then $Z \equiv Y \notin dom(\theta'_0)$ by definition of θ'_0; if $Z \in vRan(\sigma)$ then $Z \notin dom(\theta'_0)$, as we saw before.
  On the other hand, $(\mathcal{W} \setminus FV(X\ \overline{t})) \cap (FV(X\ \overline{t}) \cup FV(f\ \overline{p})) = (\mathcal{W} \setminus FV(X\ \overline{t}) \cap FV(X\ \overline{t})) \cup (\mathcal{W} \setminus FV(X\ \overline{t}) \cap FV(f\ \overline{p})) = \emptyset \cup \emptyset = \emptyset$, because $FV(f\ \overline{p})$ are part of the fresh variant. So, if $Y \in \mathcal{W} \setminus FV(X\ \overline{t})$, then $Y \notin dom(\sigma) \subseteq FV(X\ \overline{t}) \cup FV(f\ \overline{p})$. Now if $Y \in dom(\theta'_0)$ then $Y\theta \equiv Y\theta'_0$ (by definition of θ'_0), $Y\theta'_0 \equiv Y\theta'$ (as $Y \in dom(\theta'_0)$), $Y\theta' \equiv Y\sigma\theta'$ (as $Y \notin dom(\sigma)$). If $Y \in dom(\theta'_1)$, $Y\theta \equiv Y(\theta \uplus \gamma)$ (as $Y \in \mathcal{W} \setminus FV(X\ \overline{t})$ implies it does not appear in the fresh instance), $Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1$ (as $\sigma\theta'_1 = \theta \uplus \gamma$), $Y\sigma\theta'_1 \equiv Y\theta'_1$ (as $Y \notin dom(\sigma)$), $Y\theta'_1 \equiv Y\theta'$ (as $Y \in dom(\theta'_1)$) and $Y\theta' \equiv Y\sigma\theta'$ (as $Y \notin dom(\sigma)$). And if $Y \notin (dom(\theta'_0) \cup dom(\theta'_1))$ then $Y \notin dom(\theta')$, and as $Y \notin dom(\sigma)$ and $Y\theta \equiv Y(\theta \uplus \gamma)$, then $Y\theta \equiv Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1 \equiv Y \equiv Y\sigma\theta'$.
- Condition *iii.1)* : $dom(\theta') \cap \mathcal{B} = \emptyset$. Remember $\theta' = \theta'_0 \uplus \theta'_1$:
  - $dom(\theta'_0) \cap \mathcal{B} = \emptyset$: Given $Y \in dom(\theta'_0)$ then $Y \in dom(\theta)$ by definition of θ'_0, and so $Y \notin \mathcal{B}$, because $dom(\theta) \cap \mathcal{B} = \emptyset$ by hypothesis.
  - $dom(\theta'_1) \cap \mathcal{B} = \emptyset$: As σ is an mgu and $\sigma \lesssim \theta \uplus \gamma$, then $dom(\sigma) \subseteq dom(\theta \uplus \gamma)$. Given $Z \in \mathcal{B}$ then $Z \notin dom(\theta)$, as $dom(\theta) \cap \mathcal{B} = \emptyset$ by hypothesis, and $Z \notin dom(\gamma) \subseteq FV(f\ \overline{p} \rightarrow s)$ which are fresh, so

$Z \notin dom(\sigma)$. But then, as $\sigma\theta'_1 = \theta \uplus \gamma$, $Z \equiv Z(\theta \uplus \gamma) \equiv Z\sigma\theta'_1 \equiv Z\theta'_1$, so $Z \notin dom(\theta'_1)$.

- Condition *iii.2)* : $vRan(\theta') \cap \mathcal{B} = \emptyset$. Remember $\theta' = \theta'_0 \uplus \theta'_1$:
  - $vRan(\theta'_0) \cap \mathcal{B} = \emptyset$: Given $Y \in dom(\theta'_0)$ then $Y\theta'_0 \equiv Y\theta$ by definition of θ'_0. As $vRan(\theta) \cap \mathcal{B} = \emptyset$ by hypothesis then it must happen $var(Y\theta) \cap \mathcal{B} = \emptyset$, so $var(Y\theta'_0) \cap \mathcal{B} = \emptyset$.
  - $vRan(\theta'_1) \cap \mathcal{B} = \emptyset$: As $\sigma\theta'_1 = \theta \uplus \gamma$ then we can assume $dom(\theta'_1) \subseteq vRan(\sigma) \cup (dom(\theta \uplus \gamma) \setminus dom(\sigma))$.
    - Let $X \in dom(\theta'_1) \cap vRan(\sigma)$ be such that $X\theta'_1 \equiv r[Z]$ with $Z \in \mathcal{B}$. We will see that this $Z \in \mathcal{B}$ cannot appear in $X\theta'_1$ without leading to contradiction. The intuition is, as $vRan(\theta) \cap \mathcal{B} = \emptyset$ and $vRan(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$, then every $Z \in \mathcal{B}$ must come from anappearance in e of the same variable, transmitted to e' by the matching substitution γ, and so transmitted to e'' by σ.
      As $X \in vRan(\sigma)$ then there must exists $Y \in dom(\sigma)$ such that $Y \longmapsto^{\sigma} s_1[X]_p \longmapsto^{\theta'_1} s_2[r[Z]]_p$. But as $\sigma\theta'_1 = \theta \uplus \gamma$ then $Y \longmapsto^{\theta \uplus \gamma} s_2[r[Z]]_p$. Then, $Z \in vRan(\theta \uplus \gamma)$, but $Z \in \mathcal{B}, vRan(\theta) \cap \mathcal{B} = \emptyset, vRan(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset, dom(\gamma) \subseteq FV(f\ \overline{p} \to s)$, so it must happen $Z \in vRan(\gamma|_{FV(\overline{p})})$, and as a consequence $Y \in FV(\overline{p})$. Let $o \in \mathcal{O}(f\ \overline{p})$ (set of positions in $f\ \overline{p}$) be such that $f\ \overline{p}|_o \equiv Y$, then:
      - $((X\ \overline{t})\sigma)|_o \equiv ((f\ \overline{p})\sigma)|_o \equiv ((f\ \overline{p})|_o)\sigma \equiv Y\sigma \equiv s_1[X]_p$.
      - As $X\ \overline{t} \notin dom(\gamma)$, which are the fresh variables of the variant of the program rule, $((X\ \overline{t})\theta)|_o \equiv ((X\ \overline{t})(\theta \uplus \gamma))|_o \equiv ((f\ \overline{p})(\theta \uplus \gamma))|_o \equiv ((f\ \overline{p})|_o)(\theta \uplus \gamma) \equiv Y(\theta \uplus \gamma) \equiv s_2[r[Z]]_p$

      So, as $X \in dom(\theta'_1)$ then $X \notin \mathcal{B}$ and $Z \in \mathcal{B}$ has been introduced by θ, but this is impossible as $vRan(\theta) \cap \mathcal{B} = \emptyset$.
    - Let $Y \in dom(\theta) \setminus dom(\sigma)$ be. Then $Y\theta \equiv Y(\theta \uplus \gamma)$ (as $Y \in dom(\theta)$), $Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1$ (as $\sigma\theta'_1 = \theta \uplus \gamma$), $Y\sigma\theta'_1 \equiv Y\theta'_1$ (as $Y \notin dom(\sigma)$. But then no var in $\mathcal{B}$ can appear in $Y\theta'_1 \equiv Y\theta$ as $(dom(\theta) \cup vRan(\theta)) \cap \mathcal{B} = \emptyset$.
    - Let $Y \in dom(\gamma) \setminus dom(\sigma)$ be. Then $Y\gamma \equiv Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1 \equiv Y\theta'_1$, reasoning like in the previous case. As $dom(\gamma) \subseteq FV(f\ \overline{p} \to s)$ it can happen:
      - $Y \notin FV(f\ \overline{p})$: Then no var in $\mathcal{B}$ can appear in $Y\gamma$ because $vRan(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$ by the hypothesis.
      - $Y \in FV(f\ \overline{p})$: Let $Z \in \mathcal{B}$ appearing in $Y\gamma$, then Z appears in $f\ \overline{t}$ because $(f\ \overline{p})\gamma \equiv (f\ \overline{t})\theta$ and $vRan(\theta) \cap \mathcal{B} = \emptyset$. So it must happen $Y \in dom(\sigma)$ because otherwise σ could not be a unifier of $(f\ \overline{t})$ and $(f\ \overline{p})$ as Y is part of the fresh instance and so it cannot belong to $\mathcal{B}$. But this is a contradiction so this case is impossible.

2. $e \equiv f\ \overline{t}$. Then we can proceed in a similar way as we did in the previous case, but usin (Narr) instead of (VAct).

Contx Then we have $e \equiv \mathcal{C}[s]$. By hypothesis $(dom(\theta) \cup vRan(\theta)) \cap \mathcal{B} = \emptyset$ and $BV(\mathcal{C}[s]) \subseteq \mathcal{B}$, so by lemma 11 $e\theta \equiv (\mathcal{C}[s])\theta \equiv \mathcal{C}\theta[s\theta]$, and the step was

$$e\theta \equiv \mathcal{C}\theta[s\theta] \rightarrow^{l} \mathcal{C}\theta[s'] \equiv e', \text{ because } s\theta \rightarrow^{l} s'$$

Then we know that the lemma holds for $s\theta \rightarrow^{l} s'$, by the proof of the other cases, so taking $\mathcal{W}' = \mathcal{W} \cup FV(s)$ and $\mathcal{B}' = \mathcal{B}$ (as $BV(s) \subseteq BV(\mathcal{C}[s])$) we can do $s \rightsquigarrow^{l}{}_{\sigma_2} s''$ for some θ'_2 under the conditions stipulated. Now we can put this step into (Contx) to do:

$$e \equiv \mathcal{C}[s] \rightsquigarrow^{l}{}_{\sigma_2} \mathcal{C}\sigma_2[s''] \equiv e'' \text{ taking } \sigma = \sigma_2 \text{ and } \theta' = \theta'_2$$

because:

- If $s \rightsquigarrow^{l}{}_{\sigma_2} s''$ was a (Narr) or (VAct) step which lifts a (Fapp) step that uses the fresh variant $(f\ \overline{p} \rightarrow r) \in \mathcal{P}$ and adjusts with $\gamma \in PSubst$, then:
  - $dom(\sigma_2) \cap BV(\mathcal{C}) = \emptyset$: As $\sigma_2 = mgu(s, f\ \overline{p})$ then $dom(\sigma_2) \subseteq FV(s) \cup FV(f\ \overline{p})$. As $\sigma_2 \lesssim \theta \uplus \gamma$ and it is an mgu then $dom(\sigma_2) \subseteq dom(\theta \uplus \gamma)$. If $X \in FV(s) \cap dom(\sigma_2)$ then $X \notin dom(\gamma) \subseteq FV(f\ \overline{p} \rightarrow r)$, so it must happen $X \in dom(\theta)$; but then $X \notin BV(\mathcal{C})$ because $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ by the variable convention.
    Otherwise it could happen $X \in FV(f\ \overline{p}) \cap dom(\sigma_2)$, then X appears in the fresh variant and so it cannot appear in $\mathcal{C}$.
  - $vRan(\sigma_2|_{\setminus var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$: As $dom(\sigma_2) \subseteq FV(s) \cup FV(f\ \overline{p})$ then $vRan(\sigma_2|_{\setminus var(\overline{p})}) = vRan(\sigma_2|_{FV(s)})$. But as $\sigma_2 = mgu(s, f\ \overline{p})$ then $vRan(\sigma|_{FV(s)}) \subseteq FV(f\ \overline{p})$, which are part of the fresh variant, so every variable in $vRan(\sigma_2|_{\setminus var(\overline{p})})$ is fresh and so cannot appear in $\mathcal{C}$.
- If $s \rightsquigarrow^{l}{}_{\sigma_2} s''$ was a (VBind) step then:
  - $dom(\sigma_2) \cap BV(\mathcal{C}) = \emptyset$: As $dom(\sigma_2) = \overline{Z} \subseteq dom(\theta)$, and $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ as we saw before.
  - $vRan(\sigma_2) \cap BV(\mathcal{C}) = \emptyset$: Because $vRan(\sigma_2)$ only contains fresh patterns.

Then the conditions in lemma 6 hold:

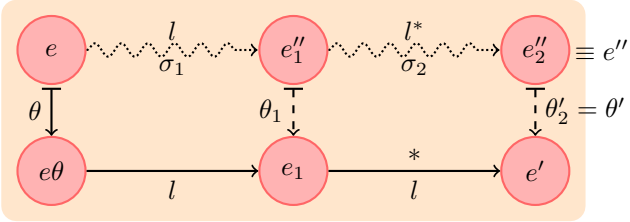
- Condition *ii)* : $\sigma\theta' = \theta[\mathcal{W}]$: Because $\mathcal{W} \subseteq \mathcal{W}'$, and $\sigma_2\theta'_2 = \theta[\mathcal{W}']$, by the proof of the other cases.
- Condition *i)* : $e''\theta' \equiv e'$: As $BV(\mathcal{C}\sigma_2) = BV(\mathcal{C})$, by the variable convention, $BV(\mathcal{C}) \subseteq BV(e) \subseteq BV(\mathcal{B})$, by the hypothesis, and $(dom(\theta'_2) \cup vRan(\theta'_2)) \cap \mathcal{B} = \emptyset$, by the proof of the other cases, then $(dom(\theta'_2) \cup vRan(\theta'_2)) \cap BV(\mathcal{C}\sigma_2) = \emptyset$. But then:

$$e''\theta' \equiv (\mathcal{C}\sigma_2[s''])\theta'_2 \equiv \underbrace{\mathcal{C}\sigma_2\theta'_2}_{\mathcal{C}\theta}[\underbrace{s''\theta'_2}_{s'}] \equiv e'$$

  Because we have $s''\theta'_2 \equiv s'$, by the proof of the other cases, and because $FV(\mathcal{C}) \subseteq FV(e) \subseteq \mathcal{W}$ and $\sigma_2\theta'_2 = \theta[\mathcal{W}]$, as we saw in the previous case (remember $\sigma = \sigma_2$ and $\theta' = \theta'_2$).

– Condition *iii)* : $(dom(\theta') \cup vRan(\theta')) \cap \mathcal{B} = \emptyset$: Because $\theta' = \theta'_2$ and the proof of the other cases.

Now we will prove the lemma for any number of steps proceeding by induction over the number n of steps of the derivation $e\theta \rightarrow^{l^n} e'$. The base case where $n = 0$ is straightforward, as then we have $e\theta \rightarrow^{l^0} e\theta \equiv e'$ so we can do $e \rightsquigarrow^{l^0}_{\epsilon} e \equiv e''$, so $\sigma = \epsilon$ and taking $\theta' = \theta$ the lemma holds. In the inductive step we have $e\theta \rightarrow^{l} e_1 \rightarrow^{l^*} e'$, and we will try to build the following diagram:



By the proof for one step we have $e \rightsquigarrow^{l}_{\sigma_1} e''_1$ and $\theta'_1 \in PSubst$ under the conditions stipulated. In order to deal with the IH we define the sets $\mathcal{B}_1 = \mathcal{B} \cup BV(e_1)$ and $\mathcal{W}_1 = (\mathcal{W} \setminus dom(\sigma_1)) \cup vRan(\sigma_1) \cup vE$, where vE is the set of extra variables in the fresh variant $f\ \overline{p} \rightarrow s$ used in $e \rightsquigarrow^{l}_{\sigma_1} e''_1$, if it was a (Narr) or (VAct) step; $vE = \overline{U} = vRan(\sigma_1)$, if it was a (VBind) step; or vE is empty otherwise. We also define $\theta_1 = \theta'_1|_{\mathcal{W}_1}$. Then:

– $FV(e''_1) \cup dom(\theta_1) \subseteq \mathcal{W}_1$: We have $dom(\theta_1) \subseteq \mathcal{W}_1$ by definition of θ_1. On the other hand we can prove $FV(e''_1) \subseteq \mathcal{W}_1$ just reasoning as we did in the first order version of this lemma, just adding a case for $X \notin FV(e)$ and X introduced by a (VBind) step, in which $X \in vE$, and hence $X \in \mathcal{W}_1$.
– $e''_1\theta_1 \equiv e_1$: Because as we have seen, $FV(e''_1) \subseteq \mathcal{W}_1$, and so $e''_1\theta_1 \equiv e''_1\theta'_1|_{\mathcal{W}_1} \equiv e''_1\theta'_1 \equiv e_1$, by the proof for one step.
– $BV(e''_1) \subseteq \mathcal{B}_1$: As $\theta'_1 \in PSubst$, $e''_1\theta'_1 \equiv e_1$ and no $PSubst$ can introduce any binding then $BV(e_1) = BV(e''_1)$. But $\mathcal{B}_1 = \mathcal{B} \cup BV(e_1)$, so $BV(e''_1) = BV(e_1) \subseteq \mathcal{B}_1$.
– $(dom(\theta_1) \cup vRan(\theta_1)) \cap \mathcal{B}_1 = \emptyset$: As $\theta'_1 \in PSubst$, $e''_1\theta'_1 \equiv e_1$ and no $PSubst$ can introduce any binding then $BV(e_1) = BV(e''_1)$. Then it can happen:
  a) $BV(e''_1) \subseteq BV(e)$: Then $\mathcal{B} = \mathcal{B}_1$, as $BV(e_1) = BV(e''_1) \subseteq BV(e) \subseteq \mathcal{B}$ by hypothesis. Then, as $(dom(\theta'_1) \cup vRan(\theta'_1)) \cap \mathcal{B} = \emptyset$ by the proof for one step, then $(dom(\theta'_1) \cup vRan(\theta'_1)) \cap \mathcal{B}_1 = \emptyset$, and so $(dom(\theta_1) \cup vRan(\theta_1)) \cap \mathcal{B}_1 = \emptyset$, because $\theta_1 = \theta'_1|_{\mathcal{W}_1}$ and so its domain and variable range is smaller than the domain of θ'_1.
  b) $BV(e''_1) \supset BV(e)$: Then $e \rightsquigarrow^{l}_{\sigma_1} e''_1$ must have been a (LetIn) step and so $\sigma = \epsilon$ and $\theta'_1 = \theta$. As the new bounded variable Z is fresh wrt θ then it is also fresh for $\theta'_1 = \theta$, and so $\mathcal{B}_1 = \mathcal{B} \cup \{Z\}$ has no intersection with $dom(\theta'_1) \cup vRan(\theta'_1)$ nor with $dom(\theta_1) \cup vRan(\theta_1)$, which is smaller.
– For each instance of program rule $R\mu \in [\mathcal{P}]$ used in an (Fapp) step it happens $vRan(\mu|_{vExtra(R)}) \cap \mathcal{B}_1 = \emptyset$. As we have seen in the previuos case either $\mathcal{B}_1 = \mathcal{B}$ or $\mathcal{B}_1 = \mathcal{B} \cup \{Z\}$ for some Z fresh, so we can assume without loss of generality that for any of those μ we have $Z \notin vRan(\mu|_{vExtra(R)})$.

– $\sigma_1\theta_1 = \theta[\mathcal{W}]$: It is enough to see that $\sigma_1\theta_1 = \sigma_1\theta'_1[\mathcal{W}]$, because we have $\sigma_1\theta'_1 = \theta[\mathcal{W}]$ by the proof for one step, and this is true because given $X \in \mathcal{W}$:
  a) If $X \in dom(\sigma_1)$ then $FV(X\sigma_1) \subseteq vRan(\sigma_1) \subseteq \mathcal{W}_1$, so as $\theta_1 = \theta'_1|_{\mathcal{W}_1}$ then $X\sigma_1\theta_1 \equiv X\sigma_1\theta'_1|_{\mathcal{W}_1} \equiv X\sigma_1\theta'_1$.
  b) If $X \in \mathcal{W}\backslash dom(\sigma_1)$ then $X \in \mathcal{W}_1$ by definition, and so $X\sigma_1\theta_1 \equiv X\theta_1$ (as $X \notin dom(\sigma_1)$), $X\theta_1 \equiv X\theta'_1|_{\mathcal{W}_1} \equiv X\theta'_1$ (as $X \in \mathcal{W}_1$), and $X\theta'_1 \equiv X\sigma\theta'_1$ (as $X \notin dom(\sigma_1)$).

So we have $e''_1\theta_1 \equiv e_1$ and $e_1 \rightarrow^{l^*} e'$, but then we can apply the induction hypothesis to $e''_1\theta_1 \rightarrow^{l^*} e'$ using $\mathcal{W}_1$ and $\mathcal{B}_1$, which fulfil the hypothesis of the lemma, as we have seen. Then we get $e''_1 \rightsquigarrow^{l^*}_{\sigma_2} e''_2$ and $\theta'_2 \in PSubst$ under the conditions stipulated. But then we have:

$$e \rightsquigarrow^{l}_{\sigma_1} e''_1 \rightsquigarrow^{l^*}_{\sigma_2} e''_2 \text{ taking } e'' \equiv e''_2,\ \sigma = \sigma_1\sigma_2 \text{ and } \theta' = \theta'_2$$

for which we can prove:

– Condition *i)* : $e''\theta' \equiv e'$: As $e''\theta' \equiv e''_2\theta'_2 \equiv e'$ by IH.
– Condition *ii)* : $\sigma\theta' = \theta[\mathcal{W}]$: That is, $\sigma_1\sigma_2\theta'_2 = \theta[\mathcal{W}]$. As we have $\sigma_1\theta_1 = \theta[\mathcal{W}]$, as we saw before, all that is left is proving $\sigma_1\sigma_2\theta'_2 = \sigma_1\theta_1[\mathcal{W}]$, which happens because given $X \in \mathcal{W}$:
  a) If $X \in dom(\sigma_1)$ then $FV(X\sigma_1) \subseteq vRan(\sigma_1) \subseteq \mathcal{W}_1$, so as $\sigma_2\theta'_2 = \theta_1[\mathcal{W}_1]$ by IH, then $(X\sigma_1)\sigma_2\theta'_2 \equiv (X\sigma_1)\theta_1$.
  b) If $X \in \mathcal{W}\backslash dom(\sigma_1)$ then $X \in \mathcal{W}_1$ by definition, and so, as $\sigma_2\theta'_2 = \theta_1[\mathcal{W}_1]$ by IH, then $X\sigma_1\sigma_2\theta'_2 \equiv X\sigma_2\theta'_2$ (as $X \notin dom(\sigma_1)$), $X\sigma_2\theta'_2 \equiv X\theta_1$ (as $X \in \mathcal{W}_1$), $X\theta_1 \equiv X\sigma_1\theta_1$ (as $X \notin dom(\sigma_1)$).
– Condition *iii)* : $(dom(\theta') \cup vRan(\theta')) \cap \mathcal{B} = \emptyset$: That is $(dom(\theta'_2) \cup vRan(\theta'_2)) \cap \mathcal{B} = \emptyset$, which happens as $(dom(\theta'_2) \cup vRan(\theta'_2)) \cap \mathcal{B}_1 = \emptyset$ by IH and $\mathcal{B} \subseteq \mathcal{B}_1$.

Proof (For theorem 8). Applying lemma 6 to $e\theta|_{FV(e)} \rightarrow^{l^*} e'$ with $\mathcal{W} = FV(e)$ and $\mathcal{B} = BV(e)$, as $e\theta|_{FV(e)} \equiv e\theta$.

5 A case of study: correctness of bubbling

Proof (For Theorem 9, Correctness of bubbling). The proof uses the following easy lemma about semantics of ?, which justifies also the equation $[\![\mathcal{C}[e_1]?\mathcal{C}[e_2]]\!] = [\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!]$ stated in the Theor. 9.

Lemma 17. $[\![e_1?e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$, *for any* $e_1, e_2 \in LExp_\perp$

Proof. We must prove $e_1?e_2 \twoheadrightarrow t \Leftrightarrow e_1 \twoheadrightarrow t \vee e_2 \twoheadrightarrow t$. Both implications are straigtforward using the rules of HOCRWL. For instance, for $\Rightarrow$, assume $e_1?e_2 \twoheadrightarrow t$. The derivation must use the HOCRWL rule (OR) and take the form

$$\frac{e_1 \twoheadrightarrow s \quad s \twoheadrightarrow t}{e_1?e_2 \twoheadrightarrow t}$$

if the rule $X?\ Y \rightarrow X$ of ? has been used in (OR), or a similar form with $e_2 \twoheadrightarrow s$ if $X?\ Y \rightarrow Y$ was used instead. But $e_1 \twoheadrightarrow s \quad s \twoheadrightarrow t$ implies $e_1 \twoheadrightarrow t$, and similar for e_2.

Coming back to the proof of Th. 9, we reason by induction on the number k of *let*'s occurring in $\mathcal{C}[e_1?e_2]$.

- $k = 0$: Since there is no *let* in $e_1?e_2$, we can apply Theor. 1 to obtain:

$$\begin{array}{ll}
[\![\mathcal{C}[e_1?e_2]]\!] & = \textit{(by Theor. 1)} \\
\bigcup_{t\in[\![e_1?e_2]\!]}[\![\mathcal{C}[t]]\!] & = \textit{(by Lemma 17)} \\
\bigcup_{t\in([\![\mathcal{C}[e_1]]\!]\cup[\![\mathcal{C}[e_2]]\!])}[\![\mathcal{C}[t]]\!] & = \textit{(set operations)} \\
\bigcup_{t\in[\![\mathcal{C}[e_1]]\!]}[\![\mathcal{C}[t]]\!] \cup \bigcup_{t\in[\![\mathcal{C}[e_2]]\!]}[\![\mathcal{C}[t]]\!] & = \textit{(by Theor. 1)} \\
[\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!] & = \textit{(by Lemma 17)} \\
[\![\mathcal{C}[e_1]?\mathcal{C}[e_2]]\!] &
\end{array}$$

- $k > 0$: We reason by induction on the structure of $\mathcal{C}$.

– $\mathcal{C} \equiv [\ ]$: the result is trivial.

– $\mathcal{C} \equiv \mathcal{C}'\ e$: then

$$\begin{array}{ll}
[\![\mathcal{C}[e_1?e_2]]\!] & = \\
[\![\mathcal{C}'[e_1?e_2]\ e]\!] & = \textit{(by Theor. 2)} \\
\bigcup_{t\in[\![\mathcal{C}'[e_1?e_2]]\!]}[\![t\ e]\!] & = \textit{(by IH on } \mathcal{C}'\textit{)} \\
\bigcup_{t\in[\![\mathcal{C}'[e_1]?\mathcal{C}'[e_2]]\!]}[\![t\ e]\!] & = \textit{(by Lemma 17)} \\
\bigcup_{t\in([\![\mathcal{C}'[e_1]]\!]\cup[\![\mathcal{C}'[e_2]]\!])}[\![t\ e]\!] & = \textit{(set operations)} \\
\bigcup_{t\in[\![\mathcal{C}'[e_1]]\!]}[\![t\ e]\!] \cup \bigcup_{t\in[\![\mathcal{C}'[e_2]]\!]}[\![t\ e]\!] & = \textit{(by Theor. 2)} \\
[\![\mathcal{C}'[e_1]e]\!] \cup [\![\mathcal{C}'[e_2]e]\!] & = \\
[\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!] & = \textit{(by Lemma 17)} \\
[\![\mathcal{C}[e_1]?\mathcal{C}[e_2]]\!] &
\end{array}$$

– $\mathcal{C} \equiv e\ \mathcal{C}'$: very similar to the previous one

– $\mathcal{C} \equiv \textit{let } x = e \textit{ in } \mathcal{C}'$: then

$$\begin{array}{ll}
[\![\mathcal{C}[e_1?e_2]]\!] & = \\
[\![\textit{let } x{=}e \textit{ in } \mathcal{C}'[e_1?e_2]]\!] & = \textit{(by Theor. 2,}\sigma \equiv \{x/t\}\textit{)} \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'[e_1?e_2]\sigma]\!] & = \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_1\sigma?e_2\sigma]]\!] & = \textit{(by IH on } k\textit{, that decreases)} \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_1\sigma]?\mathcal{C}'\sigma[e_2\sigma]]\!] & = \textit{(by Lemma 17)} \\
\bigcup_{t\in[\![e]\!]}([\![\mathcal{C}'\sigma[e_1\sigma]]\!] \cup [\![\mathcal{C}'\sigma[e_2\sigma]]\!]) & = \textit{(set operations)} \\
\bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_1\sigma]]\!] \cup \bigcup_{t\in[\![e]\!]}[\![\mathcal{C}'\sigma[e_2\sigma]]\!] & = \textit{(by Theor. 2)} \\
[\![\textit{let } x{=}e \textit{ in } \mathcal{C}'[e_1]]\!] \cup [\![\textit{let } x{=}e \textit{ in } \mathcal{C}'[e_2]]\!] & = \\
[\![\mathcal{C}[e_1]]\!] \cup [\![\mathcal{C}[e_2]]\!] & = \textit{(by Lemma 17)} \\
[\![\mathcal{C}[e_1]?\mathcal{C}[e_2]]\!] &
\end{array}$$

– $\mathcal{C} \equiv \textit{let } x = \mathcal{C}' \textit{ in } e$: very similar to the previous case

This ends the proof. It is interesting to observe that most of it consists of direct calculations with denotation of expressions, in the form of chains of equalities of denotations. We find this methodology quite appealing.

6 Translation to first order

Proof (For Proposition 2). Let us consider an expression $e = fo(e_{ho})$ (for some HO expression e_{ho}) and e' resulting from e by reducing one of its @-calls. We

want to prove $[\![e]\!] = [\![e']\!]$. The extension to consider any number of reductions of calls to @ in e is a straightforward induction on such a number.

The situation can be reflected by considering $e = C\ [@(h_n(e_1, \ldots, e_n), e_{n+1})]$ and two possible cases for e', depending on the two possible forms of the @-rule used:

- if $@(h_n(X_1, \ldots, X_n), Y) \to h_{n+1}(X_1, \ldots, X_n, Y) \in \mathcal{P}_{fo}$, with $h_n, h_{n+1} \in CS_{fo}$, then $e' = C\ [h_{n+1}(e_1, \ldots, e_n, e_{n+1})]$. It is trivial to prove that

$$[\![@(h_n(e_1, \ldots, e_n), e_{n+1})]\!] = [\![h_{n+1}(e_1, \ldots, e_n, e_{n+1})]\!]$$

  (because this @-rule is in fact the only applicable). Now, by Th. 1 we have

$$[\![e]\!] = [\![C\ [@(h_n(e_1, \ldots, e_n), e_{n+1})]]\!] = \bigcup_{t \in [\![@(h_n(e_1,\ldots,e_n),e_{n+1})]\!]} C\ [t] = \bigcup_{t \in [\![h_{n+1}(e_1,\ldots,e_n,e_{n+1})]\!]} C\ [t] = [\![C\ [h_{n+1}(e_1, \ldots, e_n, e_{n+1})]]\!] = [\![e']\!]$$

  .
- if $@(h_n(X_1, \ldots, X_n), Y) \to h(X_1, \ldots, X_n, Y) \in \mathcal{P}_{fo}$, with $h_n \in CS_{fo}$ and $h \in CS_{fo}$ (henceforth $h \in FS^n$), then $e' = C\ [h(e_1, \ldots, e_n, e_{n+1})]$. As before $[\![e]\!] = [\![e']\!]$, and by Th. 1 we obtain the result in a similar way to the previous case.

Lemma 18. $(fo(e))[X/fo(t)] = fo(e[X/t])$.

Proof. An easy induction over the structure of e.

Proof (For 10, Theorem Adequacy of HO-to-FO translation).

For the correctness of the transformation it is easier to use the alternative version of $HOCRWL_{let}$ of Fig. 4. Using this calculus we are in fact proving a generalized version of the result because it is proved for let-expressions instead of standard-expressions. So, the reformulation of the Theorem becomes:

Let $\mathcal{P}$ be a $HOCRWL_{let}$-program, $e \in LExp_{\perp}$, $t \in HOPat_{\perp}$ and $\mathcal{P}_{fo}$, $fo(e) \in LExp_{fo,\perp}$, $fo(t) \in CTerm_{\perp}$ the corresponding transformed $CRWL_{let}$-program, expression and pattern respectively. Then:

$$\mathcal{P} \vdash_{HOCRWL'_{let}} e \twoheadrightarrow t \Leftrightarrow \mathcal{P}_{fo} \vdash_{CRWL_{let}} fo(e) \twoheadrightarrow fo(t)\ \downarrow_{@}$$

We reason both implications separately:

($\Rightarrow$) We proceed by induction on the length l of the proof for $\mathcal{P} \vdash_{HOCRWL_{let}} e \twoheadrightarrow t$

$l = 0$ The cases **(B)**, **(RR)** and **(DC)** with $c \in DC^0$ are trivial.

$l \Rightarrow l + 1$ For the sake of simplicity and using the Prop. 2 when we write $e \twoheadrightarrow fo(t)$ we will understand $e \twoheadrightarrow fo(t)\ \downarrow_{@}$. We have the following cases:

(DC) the proof will have the form (we reason with only two arguments for simplicity, but the extension to more arguments is direct):

$$\frac{e_1 \twoheadrightarrow t_1 \quad e_2 \twoheadrightarrow t_2}{h\ e_1\ e_2 \twoheadrightarrow h\ t_1\ t_2} \qquad h \in \Sigma, \text{ if } h\ t_1\ t_2 \text{ is a partial pattern}$$

We have $fo(h\ e_1\ e_2) = @(@(h_0, fo(e_1)), fo(e_2))$ but, by proposition 2 we can work with the equivalent expression $h_2(fo(e_1), fo(e_2))$. On the other hand $fo(h\ t_1\ t_2) = h_2(fo(t_1), fo(t_2))$. In $CRWL_{let}$ we can build:

$$\dfrac{\dfrac{}{fo(e_1) \twoheadrightarrow fo(t_1)}\ i.h. \quad \dfrac{}{fo(e_2) \twoheadrightarrow fo(t_2)}\ i.h. \quad \dfrac{}{h_2(fo(t_1), fo(t_2)) \twoheadrightarrow h_2(fo(t_1), fo(t_2))}\ DC^*}{h_2(fo(e_1), fo(e_2)) \twoheadrightarrow h_2(fo(t_1), fo(t_2))}\ DC$$

(OR) Now we have a proof like:

$$\dfrac{e_1 \twoheadrightarrow t_1\ e_2 \twoheadrightarrow t_2 \quad r \twoheadrightarrow t}{f\ e_1\ e_2 \twoheadrightarrow t} \qquad \begin{array}{l}\text{if } t \text{ is a partial pattern} \\ (f\ t_1\ t_2 \to r) \in [\mathcal{P}]_\bot\end{array}$$

Again we work with $fo(f\ e_1\ e_2) = f(fo(e_1), fo(e_2))$ and can build the proof:

$$\dfrac{\dfrac{}{fo(e_1) \twoheadrightarrow fo(t_1)}\ i.h. \quad \dfrac{}{fo(e_2) \twoheadrightarrow fo(t_1)}\ i.h. \quad \dfrac{}{fo(r) \twoheadrightarrow fo(t)}\ i.h.}{f(fo(e_1), fo(e_2)) \twoheadrightarrow fo(t)}\ OR$$

that is performed using the instance $(f(fo(t_1), fo(t_2)) \to fo(r)) \in [\mathcal{P}_{fo}]_\bot$

(Let) The proof is:

$$\dfrac{e_1 \twoheadrightarrow t_1 \quad e_2[X/t_1] \twoheadrightarrow t}{let\ X = e_1\ in\ e_2 \twoheadrightarrow t} \qquad \text{if } t \text{ is a partial pattern}$$

We have $fo(let\ X = e_1\ in\ e_2) = (let\ X = fo(e_1)\ in\ fo(e_2))$ and the proof:

$$\dfrac{\dfrac{}{fo(e_1) \twoheadrightarrow fo(t_1)}\ i.h. \quad \dfrac{}{fo(e_2)[X/fo(t_1)] \overset{Lem.18}{=} fo(e_2[X/t_1]) \twoheadrightarrow fo(t)}\ i.h.}{let\ X = fo(e_1)\ in\ fo(e_2) \twoheadrightarrow fo(t)}\ Let$$

(Ap) We have:

$$\dfrac{e_1 \twoheadrightarrow t_1 \quad e_2 \twoheadrightarrow t_2 \quad (t_1\ t_2) \twoheadrightarrow t}{(e_1\ e_2) \twoheadrightarrow t}$$

Notice that if $\mathcal{P} \vdash_{HOCRWL'_{let}} (t_1\ t_2) \twoheadrightarrow t$ by i.h. we also have $\mathcal{P}_{fo} \vdash_{CRWL_{let}} @(fo(t_1), fo(t_2)) \twoheadrightarrow fo(t)$. This proof must be done by **(OR)**, using a rule $(@(s_1, s_2) \to r) \in \mathcal{P}_{fo}$ and $\theta \in CSubst_\bot$ in the following way:

$$\dfrac{fo(t_1) \twoheadrightarrow s_1\theta \quad fo(t_2) \twoheadrightarrow s_2\theta \quad r\theta \twoheadrightarrow fo(t)}{@(fo(t_1), fo(t_2)) \twoheadrightarrow fo(t)}\ OR$$

On the other hand, by i.h. we have $fo(e_1) \twoheadrightarrow fo(t_1)$ and then, as $fo(t_1) \twoheadrightarrow s_1\theta$ we also have $fo(e_1) \twoheadrightarrow s_1\theta$, and similarly $fo(e_2) \twoheadrightarrow s_2\theta$. Now, we use these facts for building our proof in $CRWL_{let}$. First of all $fo(e_1\ e_2) = @(fo(e_1), fo(e_2))$ and the proof will have the form:

$$\frac{fo(e_1) \twoheadrightarrow s_1\theta \quad fo(e_2) \twoheadrightarrow s_2\theta \quad r\theta \twoheadrightarrow fo(t)}{@(fo(e_1), fo(e_2)) \twoheadrightarrow fo(t)}\ OR$$

Using the same function rule and the same c-susbtitution.

($\Leftarrow$) We proceed by induction on the length l of the proof for $\mathcal{P}_{fo} \vdash_{CRWL_{let}} fo(e) \twoheadrightarrow fo(t)$:

$l = 0$ The cases **(B)**, **(RR)** or **(DC)** with $c \in CS^0$ are trivial.

$l \Rightarrow l + 1$ The possible proofs with length greater than one are:

* **(DC)** and **(Let)** are easy applications of i.h.
* If the proof is done by **(OR)** it can have two forms depending on the function rule applied: it can belong to $\mathcal{P}_@$ or come from a rule of the original program $\mathcal{P}$:
  - For the first case the proof is:

$$\frac{fo(e_1) \twoheadrightarrow fo(t_1)\theta \quad fo(e_2) \twoheadrightarrow fo(t_2)\theta \quad fo(r)\theta \twoheadrightarrow fo(t)}{@(fo(e_1), fo(e_2)) \twoheadrightarrow fo(t)}$$

    using a rule $(@(fo(t_1), fo(t_2)) \to fo(r)) \in \mathcal{P}_{fo}$ and $\theta \in CSubst_\perp$. Then $(e_1\ e_2)$ is a partial application and $(t_1\ t_2)$ is a pattern and it is easy to build the proof $(e_1\ e_2) \twoheadrightarrow (t_1\ t_2)$ in $HOCRWL'_{let}$ using i.h. and DC.
  - For the second one we have

$$\frac{fo(e_1) \twoheadrightarrow fo(t_1)\theta \ldots fo(e_n) \twoheadrightarrow fo(t_n)\theta \quad fo(r)\theta \twoheadrightarrow fo(t)}{fo(f(e_1, \ldots, e_n)) \twoheadrightarrow fo(t)}$$

    taking $(fo(f(t_1, \ldots, t_n)) \to fo(r)) \in \mathcal{P}_{fo}$ and $\theta \in CSubst_\perp$. Then it must be $(f(t_1, \ldots, t_n) \to r) \in \mathcal{P}$ and we have:

$$\frac{\overline{e_1 \twoheadrightarrow t_1\theta}\ i.h. \quad \ldots \overline{e_n \twoheadrightarrow t_n\theta}\ i.h. \quad \overline{r\theta \twoheadrightarrow t}\ i.h.}{f(e_1, \ldots, e_n) \twoheadrightarrow t}$$