Singular and Plural Functions for Functional Logic Programming: Detailed Proofs*

Adrián Riesco and J. Rodríguez-Hortalá

Technical Report SIC-9/11

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

November 2011

*Research partially supported by the Spanish projects *FAST-STAMP* (TIN2008-06622-C03-01/TIN), *DESAFIOS10* (TIN2009-14599-C03-01), *PROMETIDOS-CM* (S2009TIC-1465), and *GPD-UCM* (UCM-BSCH-GR58/08-910502).

Abstract

Modern functional logic programming (FLP) languages use non-terminating and non-confluent constructor systems (CS's) as programs in order to define non-strict non-determi-nistic functions. Two semantic alternatives have been usually considered for parameter passing with this kind of functions: call-time choice and run-time choice. While the former is the standard choice of modern FLP languages, the latter lacks some basic properties—mainly compositionality—that have prevented its use in practical FLP systems. Traditionally it has been considered that call-time choice induces a singular denotational semantics, while run-time choice induces a plural semantics. We have discovered that this latter identification is wrong when pattern matching is involved, and thus in this paper we propose two novel compositional plural semantics for CS's that are different from run-time choice.

We investigate the basic properties of our plural semantics—compositionality, polarity, monotonicity for substitutions, and a restricted form of the bubbling property for constructor systems—and the relation between them and to previous proposals, concluding that these semantics form a hierarchy in the sense of set inclusion of the set of values computed by them. Besides, we have identified a class of programs characterized by a simple syntactic criterion for which the proposed plural semantics behave the same, and a program transformation that can be used to simulate one of the proposed plural semantics by term rewriting. At the practical level, we study how to use the new expressive capabilities of these semantics for improving the declarative flavour of programs. As call-time choice is the standard semantics for FLP, it still remains the best option for many common programming patterns. Therefore we propose a language which combines call-time choice and our plural semantics, that we have implemented in the Maude system. The resulting interpreter is then employed to develop and test several significant examples showing the capabilities of the combined semantics.

Keywords: Non-deterministic functions, Semantics, Program transformation, Term rewriting, Maude

Contents

1 Introduction

Term rewriting [7] and term graph rewriting systems [42] have often been used for modeling the semantics and operational behaviour of functional logic programming (FLP) languages [15, 25, 4], a paradigm that tries to integrate into a single language the main features of lazy functional programming (FP) and logic programming (LP). In particular, the class of left-linear constructor-based term rewriting systems—or simply constructor systems (CS's)—, in which the signature is divided into two disjoint sets of constructor and function symbols, is used frequently to represent programs. There the notion of value as a term built using only constructor symbols—called constructor term or just c-term—arises naturally, and this way a term rewriting derivation from an expression to a c-term represents the reduction of that expression to one of its values in the language being modelled. This corresponds to a value-based semantic view, in which the purpose of computations is to produce values made of constructors. Besides, term graphs are used for modelling subexpression sharing, where several occurrences of the same subexpression are represented by several pointers to a single node in a term graph, resulting in a potential improvement of the time and space performance of programs. Sharing is at the core of implementations of lazy FP and FLP languages, and so several variations of term graph rewriting have also been used in formulations of the semantics of call-by-need in FP [29, 6, 41] and FLP [17, 1, 33, 30].

On the other hand, non-determinism is an expressive feature that has been used for a long time in programming [16, 27, 40] and system specification [14, 20, 8]. In both fields, one of the appeals of term rewriting is its elegant way to express non-determinism through the use of non-confluent term rewriting systems, obtaining a clean and high level representation of complex systems and programs. Non-determinism is integrated in FLP languages by means of a backtracking mechanism in the style of Prolog [49]. It is introduced by employing possibly non-terminating and non-confluent CS's as programs, thus expressing non-strict non-deterministic functions, which are one of the most distinctive features of the paradigm [23, 3, 4].[1]

The point is that this combination of non-strictness and non-determinism gives rise to several semantic alternatives [48, 28]. In particular in [48] the different language variants that result after adding non-determinism to a basic functional language were expounded, structuring the comparison as a choice among different options over several dimensions: strict/non-strict functions, angelic/demonic/erratic non-deterministic choices, and *singular/plural semantics* for parameter passing, also called *call-time choice/run-time choice* in [28]. In the present paper we assume non-strict angelic non-determinism, so we focus on the last dimension only. To do that, let us take a look at the following example.

Example 1.1. *Consider the program* $\{f(c(X)) \to d(X,X), X\ ?\ Y \to X, X\ ?\ Y \to Y\}$ *and the expression* $f(c(0\ ?\ 1))$. *From an operational perspective we have to decide when it is time to fix the values for the arguments of functions:*

- *Under a* call-time choice *semantics a value for each argument will be fixed on parameter passing and shared between every copy of that argument which arises during the computation. This corresponds to call-by-value in a strict setting and to call-by-need in a non-strict setting, in which a partial value instead of a total value is computed. So when applying the rule for* f *the two occurrences of* X *in* $d(X,X)$ *will share the same value, hence* $d(0,0)$ *and* $d(1,1)$ *are correct values for* $f(c(0\ ?\ 1))$ *in this semantics, while it is not the case either for* $d(0,1)$ *or for* $d(1,0)$.

- *On the other hand* run-time choice *corresponds to call-by-name, so the values of the arguments are fixed as they are used—i.e., as their evaluation is demanded by the matching process—and the copies of each argument created by parameter passing may evolve independently afterwards. Under this semantics not only* $d(0,0)$ *and* $d(1,1)$ *but also* $d(0,1)$ *and* $d(1,0)$ *are correct values for* $f(c(0\ ?\ 1))$.

In general, a call-time choice semantics produces less results than run-time choice. Modern functional-logic languages like Toy [36] or Curry [24] are heavily influenced by lazy functional programming and so they implement sharing in their operational mechanism, which results in call-by-need evaluation and the adoption of call-time choice. On the other hand, term rewriting is considered a standard formulation for run-time choice,[2] and is the basis for the semantics of languages like Maude [14].

But we may also see things from another perspective.

[1] Non-determinism also appears in FLP as a result of the utilization of narrowing as the fundamental operational mechanism [24] but, as usual in many works in the field, we will focus on rewriting aspects only, so our conclusions could be lifted to the narrowing case in subsequent works.

[2] In fact angelic non-strict run-time choice.

Example 1.2. *Consider again the program in Example 1.1. From a denotational perspective we have to think about the domain used to instantiate the variables of the program rules:*

- *Under a* singular semantics *variables will be instantiated with single values (which may be partial in a non-strict setting).* This is equivalent to having call-time choice parameter passing.
- *The alternative is having a* plural semantics*, in which the variables are instantiated with sets of values. Traditionally it has been considered that run-time choice has its denotational counterpart on a plural semantics, but we will see that this identification is wrong. Consider the expression* $f(c(0)\ ?\ c(1))$*, under run-time choice, that is, term rewriting, the evaluation of the subexpression* $c(0)\ ?\ c(1)$ *is needed in order to get an instance of the left-hand side of the rule for* f*. Hence a choice between* $c(0)$ *and* $c(1)$ *is performed and so neither* $d(0,1)$ *nor* $d(1,0)$ *are correct values for* $f(c(0)\ ?\ c(1))$*. Nevertheless, under a plural semantics we may consider the set* $\{c(0), c(1)\}$ *which is a subset of the set of values for* $c(0)\ ?\ c(1)$ *in which every element matches the argument pattern* $c(X)$*. Therefore, the set* $\{0,1\}$ *can be used for parameter passing obtaining a kind of "set expression"* $d(\{0,1\},\{0,1\})$ *that yields the values* $d(0,0)$*,* $d(1,1)$*,* $d(0,1)$*, and* $d(1,0)$*.*

The conclusion is clear: the traditional identification of run-time choice with a plural semantics is wrong when pattern matching is involved.

Which of these is the more suitable perspective for FLP? This problem did not appear in [48] because no pattern matching was present, nor in [28] because only call-time choice was adopted there. This fact was pointed out for the first time in [45], where the $\pi^{\alpha}CRWL$ logic—named $\pi CRWL$ in that work—was proposed as a novel formulation of a plural semantics with pattern matching. This proves that one can conceive a meaningful plural semantics that is different to run-time choice, i.e., run-time choice is not the only plural semantics we should consider. We have seen that, using the program above, the expression $f(c(0\ ?\ 1))$ has more values than the expression $f(c(0)\ ?\ c(1))$ under run-time choice although they only differ in the subexpressions $c(0\ ?\ 1)$ and $c(0)\ ?\ c(1)$, which have the same values under all three call-time choice, run-time choice, and plural semantics. That violates a fundamental property of FLP languages stating that any expression can be replaced by any other expression which could be reduced to exactly the same set of values. We will see that our plural semantics shares with *CRWL*[3] [23] (the standard logic for call-time choice[4]) a compositionality property for values that makes it more suitable than run-time choice for a value-based language like current implementations of FLP. Nevertheless run-time choice can be a good option for other kind of rewriting-based languages like Maude, in which the notion of value is not necessarily present, at least in the sense it is in FLP languages.

In this paper we have put together our previous results about plural semantics, integrating our presentation of $\pi^{\alpha}CRWL$ from [45] with a user level introduction to a Maude-based transformational prototype for $\pi^{\alpha}CRWL$ [43]. We have also included the results obtained in [44], which is devoted to the exploration of the new expressive capabilities of our plural semantics. Although our plural semantics allows an elegant encoding of some problems—in particular those with an implicit manipulation of sets of values—, call-time choice still remains the best option for many common programming patterns [23, 3]. Therefore we propose a combined semantics for a language in which the user can specify, for each function symbol, which arguments are considered "plural arguments"—thus being evaluated under our plural semantics—and which "singular arguments"—thus being evaluated under call-time choice. This semantics is precisely specified by a modification of the *CRWL* logic, which retains the important properties of *CRWL* and $\pi^{\alpha}CRWL$, like compositionality. These new features were implemented by extending our Maude prototype, and then used to develop and test several significant examples showing the expressive capabilities of the combined semantics

Apart from giving a unified and revised presentation, we have made several relevant advances. We have extended most of our results to deal with programs with extra variables, and above all, we have introduced the new plural semantics $\pi^{\beta}CRWL$ inspired by the proposal from [9]. The properties of this semantics and its relation to call-time choice, run-time choice, and $\pi^{\alpha}CRWL$ have been studied in depth and with technical accuracy. Our current implementation does not deal with extra variables because they cause an explosion in the search space when evaluated by term rewriting—we consider the development of a suitable plural narrowing mechanism that could effectively handle extra variables a possible subject of future work.

[3] **C**onstructor-based **R**e**W**riting **L**ogic.
[4] In fact angelic non-strict call-time choice.

The rest of the paper is organized as follows. Section 2 contains some technical preliminaries and notations about term rewriting systems and the *CRWL* logic. In Section 3 we introduce π^α*CRWL* and π^β*CRWL*, two variations of *CRWL* to express plural semantics, and present some of their properties, in particular compositionality. In Section 4 we study the relation between call-time choice, run-time choice, and our plural semantics, focusing on the set of values computed by each semantics and concluding that these four semantics form a hierarchy in the sense of set inclusion. We also present a class of programs characterized by a simple syntactic criterion under which our two plural semantics are equivalent, and conclude the section providing a simple program transformation that can be used to simulate π^α*CRWL* with term rewriting. Section 5 begins with a presentation of our combinations of call-time choice and plural semantics that are formalized through the $CRWL^\sigma_{\pi^\alpha}$ and $CRWL^\sigma_{\pi^\beta}$ logics, which correspond to the combination of call-time choice with π^α*CRWL* and π^β*CRWL*, respectively. Then follows a user level introduction to our Maude prototype, which implements the $CRWL^\sigma_{\pi^\alpha}$ logic, as it is based on the transformation from Section 4. The prototype is then employed to illustrate the use of the combined semantics for improving the declarative flavour of programs. This section concludes with a short sketch of the implementation of our prototype. Finally, in Section 6 we outline some possible lines of future work. For the sake of readability, some of the proofs have been moved to Appendix A, although the intutitions behind our main results have been presented in the text.

2 Preliminaries

We present in this section the main notions needed throughout the rest of the paper: Section 2.1 introduces constructor-based systems, while Section 2.2 describes the *CRWL* framework.

2.1 Constructor systems

We consider a first order signature $\Sigma = CS \uplus FS$, where CS and FS are two disjoint sets of *constructor* and defined *function* symbols respectively, all of them with associated arity. We write CS^n (FS^n resp.) for the set of constructor (function) symbols of arity $n \in \mathbb{N}$. We write $c, d, \ldots$ for constructors, $f, g, \ldots$ for functions, and $X, Y, \ldots$ for variables of a numerable set $\mathcal{V}$. The notation $\overline{o}$ stands for tuples of any kind of syntactic objects. Given a set $\mathcal{A}$ we denote by $\mathcal{A}^*$ the set of finite sequences of elements of that set. We denote the empty sequence by []. For any sequence $a_1 \ldots a_n \in \mathcal{A}^*$ and function $f : \mathcal{A} \to \{true, false\}$, we denote by $a_1 \ldots a_n \mid f$ the sequence constructed by taking in order every element from $a_1 \ldots a_n$ for which f holds. Finally, for any $1 \leq i \leq n$, $(a_1 \ldots a_n)[i]$ denotes a_i.

The set *Exp* of *expressions* is defined as $Exp \ni e ::= X \mid h(e_1, \ldots, e_n)$, where $X \in \mathcal{V}$, $h \in CS^n \cup FS^n$ and $e_1, \ldots, e_n \in Exp$. We use the symbol $\equiv$ for the syntactic equality between expressions, and in general for any syntactic construction. The set *CTerm* of *constructed terms* (or *c-terms*) is defined like *Exp*, but with h restricted to CS^n (so $CTerm \subseteq Exp$). The intended meaning is that *Exp* stands for evaluable expressions, i.e., expressions that can contain function symbols, while *CTerm* stands for data terms representing **values**. We will write $e, e', \ldots$ for expressions and $t, s, \ldots$ for c-terms. The set of variables occurring in an expression e will be denoted as $var(e)$. We will frequently use *one-hole contexts*, defined as $Cntxt \ni \mathcal{C} ::= [\,] \mid h(e_1, \ldots, \mathcal{C}, \ldots, e_n)$, with $h \in CS^n \cup FS^n$, $e_1, \ldots, e_n \in Exp$. The application of a context $\mathcal{C}$ to an expression e, written by $\mathcal{C}[e]$, is defined inductively as $[\,][e] = e$ and $h(e_1, \ldots, \mathcal{C}, \ldots, e_n)[e] = h(e_1, \ldots, \mathcal{C}[e], \ldots, e_n)$.

A position of an expression is a chain of natural numbers separated by dots that determines one of its subexpressions. Given an expression e by $\mathcal{O}(e)$ we denote the set of positions in e, which is defined as $\mathcal{O}(X) = \epsilon$; $\mathcal{O}(h(e_1, \ldots, e_n)) = \{\epsilon\} \cup \{i.o \mid i \in \{1, \ldots, n\} \wedge o \in \mathcal{O}(e_i)\}$, where $X \in \mathcal{V}$, $h \in \Sigma$, and ϵ denotes the empty or top position. We will write $o, p, q, u, v, \ldots$ for positions. By $e|_o$ we denote the subexpression of e at position $o \in \mathcal{O}(e)$, defined as $e|_\epsilon = e$; $h(e_1, \ldots, e_n)|_{i.o} = e_i|_o$. The set of variable positions in e is denoted as $\mathcal{O}_\mathcal{V}(e)$ and defined as $\mathcal{O}_\mathcal{V}(e) = \{o \in \mathcal{O}(e) \mid e|_o \in \mathcal{V}\}$.

Substitutions $\theta \in Subst$ are finite mappings $\theta : \mathcal{V} \longrightarrow Exp$, extending naturally to $\theta : Exp \longrightarrow Exp$. We write ϵ for the identity (or empty) substitution. We write $e\theta$ for the application of θ to e, and $\theta\theta'$ for the composition, defined by $e(\theta\theta') = (e\theta)\theta'$. The domain and variable range of θ are defined as $dom(\theta) = \{X \in \mathcal{V} \mid X\theta \neq X\}$ and $vran(\theta) = \bigcup_{X \in dom(\theta)} var(X\theta)$. If $dom(\theta_0) \cap dom(\theta_1) = \emptyset$, their disjoint union $\theta_0 \uplus \theta_1$ is defined by $(\theta_0 \uplus \theta_1)(X) = \theta_i(X)$, if $X \in dom(\theta_i)$ for some $i \in \{0, 1\}$; $(\theta_0 \uplus \theta_1)(X) = X$ otherwise. Given $W \subseteq \mathcal{V}$ we write $\theta|_W$ for the restriction of θ to W, and $\theta|_{\setminus D}$ is a shortcut for $\theta|_{(\mathcal{V} \setminus D)}$. We will sometimes write $\theta = \sigma[W]$ instead of $\theta|_W = \sigma|_W$. *C-substitutions* $\theta \in CSubst$ verify that

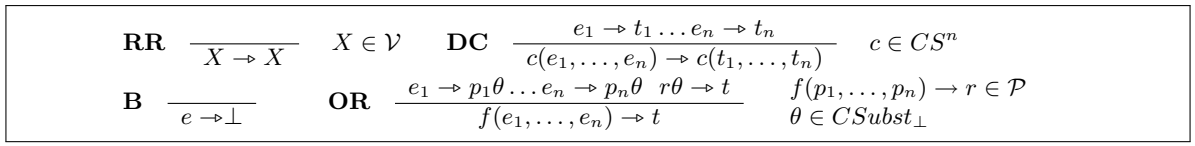

Figure 1: Rules of *CRWL*

$X\theta \in CTerm$ for all $X \in dom(\theta)$. We say that e *subsumes* e', and write $e \precsim e'$, if $e\sigma \equiv e'$ for some substitution σ.

A *constructor-based term rewriting system* (*CS*) or just *constructor system* or *program* $\mathcal{P}$ is a set of *rewrite rules* or *program rules* of the form $f(t_1,\ldots,t_n) \to r$ where $f \in FS^n$, $e \in Exp$, and $(t_1,\ldots,t_n)$ is a linear tuple of c-terms, where linearity means that variables occur only once in $(t_1,\ldots,t_n)$. Notice that we allow r to contain *extra variables*, i.e., variables not occurring in $(t_1,\ldots,t_n)$. To be precise, we say that $X \in \mathcal{V}$ is an extra variable in the rule $l \to r$ iff $X \in var(r) \setminus var(l)$, and by $vExtra(R)$ we denote the set of extra variables in a program rule R. For any program $\mathcal{P}$ the set $FS^{\mathcal{P}}$ of functions defined by $\mathcal{P}$ is $FS^{\mathcal{P}} = \{f \in FS \mid \exists(f(\overline{p}) \to r) \in \mathcal{P}\}$. We assume that every program $\mathcal{P}$ contains the rules $\{X\ ?\ Y \to X, X\ ?\ Y \to Y, \mathit{if\ true\ then}\ X \to X\}$, defining the behaviour of the infix function $? \in FS^2$ and the mixfix function $\mathit{if\ then} \in FS^2$ (used as $\mathit{if}\ e_1\ \mathit{then}\ e_2$), and that those are the only rules for that function symbols. Besides ? is right-associative, so $e_1\ ?\ e_2\ ?\ e_3 \equiv e_1\ ?\ (e_2\ ?\ e_3)$. For the sake of conciseness we will often omit these rules when presenting a program.

Given a program $\mathcal{P}$, its associated *term rewriting relation* $\to_{\mathcal{P}}$ is defined as: $\mathcal{C}[l\sigma] \to_{\mathcal{P}} \mathcal{C}[r\sigma]$ for any context $\mathcal{C}$, rule $l \to r \in \mathcal{P}$ and $\sigma \in Subst$. We write $\overset{*}{\to}_{\mathcal{P}}$ for the reflexive and transitive closure of the relation $\to_{\mathcal{P}}$. In the following, we will usually omit the reference to $\mathcal{P}$ or denote it by $\mathcal{P} \vdash e \to e'$ and $\mathcal{P} \vdash e \to^* e'$.

2.2 The CRWL framework

The *CRWL* framework [22, 23] is considered a standard formulation of call-time choice by the FLP community [25, 4]. To deal with non-strictness at the semantic level, Σ is enlarged with a new constant constructor symbol $\bot$. The sets $Exp_\bot$, $CTerm_\bot$, $Subst_\bot$, $CSubst_\bot$ of *partial expressions*, etc., are defined naturally. Our contexts will contain partial expressions from now on unless explicitly specified. Expressions, substitutions, etc. not containing $\bot$ are called *total*. Programs in *CRWL* still consist of rewrite rules with total expressions in both sides, so $\bot$ does not appear in programs. 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 $X \in \mathcal{V}$.

The semantics of a program $\mathcal{P}$ is determined in *CRWL* by means of a proof calculus able to derive *reduction statements* of the form $e \twoheadrightarrow t$, with $e \in Exp_\bot$ and $t \in CTerm_\bot$, meaning informally that t is (or approximates to) a *possible value* of e, obtained by iterated reduction of e using $\mathcal{P}$ under call-time choice. The *CRWL*-proof calculus is presented in Figure 1. Rules **RR** (restricted reflexivity) and **DC** (decomposition) are used to reduce any variable to itself, and to decompose the evaluation of constructor-rooted expressions. Rule **B** (bottom) allows us to avoid the evaluation of expressions, in order to get a non-strict semantics. Finally rule **OR** (outer reduction) expresses that to evaluate a function call we must first evaluate its arguments to get an instance of a program rule, perform parameter passing (by means of some substitution $\theta \in CSubst_\bot$), and then reduce the correspondingly instantiated right-hand side. The use of partial c-substitutions in **OR** is essential to express call-time choice, as only single partial values are used for parameter passing. Notice also that by the effect of θ in **OR**, extra variables in the right-hand side of a rule can be replaced by any partial c-term, but not by any expression as in term rewriting.

We write $\mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t$ to express that $e \twoheadrightarrow t$ is derivable in the *CRWL*-calculus using the program $\mathcal{P}$. Given a program $\mathcal{P}$, the *CRWL-denotation* of an expression $e \in Exp_\bot$ is defined as $[\![e]\!]^{sg}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t\}$. In the following, we will usually omit the reference to $\mathcal{P}$ when implied by the context.

3 Two plural semantics for constructor systems

In this section we present two semantic proposals for constructor systems that are plural in the sense described in the introduction, but at the same time are different to the run-time choice semantics induced by term rewriting. We will formalize them by means of two modifications of the *CRWL* proof calculus, that will now consider sets of partial values for parameter passing instead of single partial values. Thus only the rule **OR** should be modified. To avoid the need to extend the syntax with new constructions to represent those "set expressions" that we mentioned in the introduction, we will exploit the fact that $[\![e_1 \ ? \ e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$ for any sensible semantics—in particular each of the semantics considered in this work. Therefore the substitutions used for parameter passing will map variables to "disjunctions of values." We define the set $CSubst^?_\bot = \{\theta \in Subst_\bot \mid \forall X \in dom(\theta), \theta(X) = t_1 \ ? \ \ldots \ ? \ t_n$ such that $t_1, \ldots, t_n \in CTerm_\bot, n > 0\}$, for which $CSubst_\bot \subseteq CSubst^?_\bot \subseteq Subst_\bot$ obviously holds. The operator $? : CSubst^*_\bot \to CSubst^?_\bot$ constructs the $CSubst^?_\bot$ corresponding to a non-empty sequence of $CSubst_\bot$, and it is defined as follows:

$$?(\theta_1 \ldots \theta_n)(X) = \begin{cases} X \ ? \ \rho_1(X) \ ? \ \ldots \ ? \ \rho_m(X) & \text{if } \exists \theta_i \text{ such that } X \notin dom(\theta_i) \\ \theta_1(X) \ ? \ \ldots \ ? \ \theta_n(X) & \text{otherwise} \end{cases}$$

where $\rho_1 \ldots \rho_m = \theta_1 \ldots \theta_n \mid \lambda\theta.(X \in dom(\theta))$. This operator is overloaded to handle non-empty sets $\Theta \subseteq CSubst_\bot$ as $?\Theta = ?(\theta_1 \ldots \theta_n)$ where the sequence $\theta_1 \ldots \theta_n$ corresponds to an arbitrary reordering of the elements of Θ—for example using some standard order of terms in the line of [49].

Lemma 1. *For any $\theta_1, \ldots, \theta_n \in CSubst_\bot$, $dom(?\{\theta_1 \ldots \theta_n\}) = \bigcup_i dom(\theta_i)$.*

Proof. Simple calculations using the definition of $?\{\theta_1 \ldots \theta_n\}$, see Appendix A (page 36) for details. □

3.1 $\pi^\alpha CRWL$

Our first semantic proposal is defined by the $\pi^\alpha CRWL$-proof calculus in Figure 2. The only difference with the *CRWL* proof calculus in Figure 1 is that the rule **OR** has been replaced by $\mathbf{POR}^\alpha$ (alpha plural outer reduction), in which we may compute more than one partial value for each argument, and then use a substitution from $CSubst^?_\bot$ instead of $CSubst_\bot$ for parameter passing, achieving a plural semantics.[5] Besides, extra variables are instantiated by an arbitrary $\theta_e \in CSubst^?_\bot$ for the same reason. Just like *CRWL*, the calculus evaluates expressions in an innermost way, and avoids the use of any transitivity rule that would induce a step-wise semantics like e.g. term rewriting. The motivation for that is to get a compositional calculus in the values it computes, i.e., that the semantics of an expression would only depend on the semantics of its constituents, in a simple way—we will give a formal characterization for that in Theorem 1 below. Note that the use of partial c-terms as values is crucial to prevent innermost evaluation from making functions strict, thus losing lazy evaluation. Fortunately the rule **B** combined with the use of partial substitutions for parameter passing ensure a lazy behaviour for both $\pi^\alpha CRWL$ and *CRWL*. Therefore we could roughly describe the parameter passing of *CRWL* as call-by-partial-value, while $\pi^\alpha CRWL$ would perform call-by-set-of-partial-values.

The calculus derives *reduction statements* of the form $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$, which expresses that t is (or approximates to) a possible value for e in this semantics, under the program $\mathcal{P}$. For any $\pi^\alpha CRWL$-proof we define its *size* as the number of applications of rules of the calculus. The *$\pi^\alpha CRWL$-denotation* of an expression $e \in Exp_\bot$ under a program $\mathcal{P}$ in $\pi^\alpha CRWL$ is defined as $[\![e]\!]^{\alpha pl}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t\}$. In the following, we will usually omit the reference to $\mathcal{P}$ and αpl, and even will skip $\vdash_{\pi^\alpha CRWL}$, when it is clearly implied by the context.

Example 3.1. *Consider the program of Example 1.1, that is $\{f(c(X)) \to d(X,X)$, $X \ ? \ Y \to X$, $X \ ? \ Y \to Y\}$. The following is a $\pi^\alpha CRWL$-proof for the statement $f(c(0) \ ? \ c(1)) \twoheadrightarrow d(0,1)$ (some steps have been omitted for the sake of conciseness):*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{0 \twoheadrightarrow 0}\,\mathbf{DC}}{c(0) \twoheadrightarrow c(0)}\,\mathbf{DC} \quad \dfrac{}{c(1) \twoheadrightarrow \bot}\,\mathbf{B} \quad c(0) \twoheadrightarrow c(0)}{c(0)?c(1) \twoheadrightarrow c(0)}\,\boldsymbol{POR}^\alpha \quad c(0)?c(1) \twoheadrightarrow c(1) \quad \dfrac{0?1 \twoheadrightarrow 0 \quad 0?1 \twoheadrightarrow 1}{d(0?1, 0?1) \twoheadrightarrow d(0,1)}\,\mathbf{DC}}{f(c(0)?c(1)) \twoheadrightarrow d(0,1)}\,\boldsymbol{POR}^\alpha$$

[5] In fact angelic non-strict plural non-determinism.

$$\textbf{RR}\ \frac{}{X \twoheadrightarrow X}\quad X \in \mathcal{V} \qquad \textbf{DC}\ \frac{e_1 \twoheadrightarrow t_1 \ldots e_n \twoheadrightarrow t_n}{c(e_1,\ldots,e_n) \twoheadrightarrow c(t_1,\ldots,t_n)}\quad c \in CS^n$$

$$\textbf{B}\ \frac{}{e \twoheadrightarrow \bot} \qquad \textbf{POR}^{\alpha}\ \frac{\begin{array}{ccc} e_1 \twoheadrightarrow p_1\theta_{11} & & e_n \twoheadrightarrow p_n\theta_{n1} \\ \ldots & \ldots & \ldots \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & e_n \twoheadrightarrow p_n\theta_{nm_n} \quad r\theta \twoheadrightarrow t \end{array}}{f(e_1,\ldots,e_n) \twoheadrightarrow t}$$

if $(f(\overline{p}) \to r) \in \mathcal{P}, \forall i\ \Theta_i = \{\theta_{i1},\ldots,\theta_{im_i}\}$
$\theta = (\biguplus ?\Theta_i) \uplus \theta_e, \forall i,j\ dom(\theta_{ij}) \subseteq var(p_i), \forall i\ m_i > 0$
$dom(\theta_e) \subseteq vExtra(f(\overline{p}) \to r), \theta_e \in CSubst^{?}_{\bot}$

Figure 2: Rules of $\pi^{\alpha}CRWL$

One of the most important properties of $\pi^{\alpha}CRWL$ is compositionality, a property very close to the DET-additivity property for algebraic specifications of [28], or the referencial transparency property of [47]. This property shows that the $\pi^{\alpha}CRWL$-denotation of any expression put in a context only depends on the $\pi^{\alpha}CRWL$-denotation of that expression, and formalizes the idea that the semantics of a whole expression depends only on the semantics of its constituents, as we informally pointed above.

Theorem 1 (Compositionality of $\pi^{\alpha}CRWL$)**.** *For any program, $\mathcal{C} \in Cntxt$ and $e \in Exp_{\bot}$:*

$$[\![\mathcal{C}[e]]\!]^{\alpha pl} = \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![e]\!]^{\alpha pl}} [\![\mathcal{C}[t_1\ ?\ \ldots\ ?\ t_n]]\!]^{\alpha pl}$$

for any arrangement of the elements of $\{t_1,\ldots,t_n\}$ in $t_1\ ?\ \ldots\ ?\ t_n$. As a consequence, for any $e' \in Exp_{\bot}$:

$$[\![e]\!]^{\alpha pl} = [\![e']\!]^{\alpha pl} \text{ iff } \forall \mathcal{C} \in Cntxt. [\![\mathcal{C}[e]]\!]^{\alpha pl} = [\![\mathcal{C}[e']]\!]^{\alpha pl}$$

Proof. We have to prove that, for any $t \in CTerm_{\bot}$, if $\mathcal{C}[e] \twoheadrightarrow t$ then $\exists \{s_1,\ldots,s_n\} \subseteq [\![e]\!]^{\alpha pl}$ such that $\mathcal{C}[s_1\ ?\ \ldots\ ?\ s_n] \twoheadrightarrow t$; and conversely, that given $\{s_1,\ldots,s_n\} \subseteq [\![e]\!]^{\alpha pl}$ such that $\mathcal{C}[s_1\ ?\ \ldots\ ?\ s_n] \twoheadrightarrow t$ then $\mathcal{C}[e] \twoheadrightarrow t$. Each of these statements can be proved by induction on the size of the starting proof, see Appendix A (page 37) for details. □

Contrary to what happens to call-time choice [34, 30], we cannot have a compositionality result for single values like $[\![\mathcal{C}[e]]\!] = \bigcup_{t \in [\![e]\!]} [\![\mathcal{C}[t]]\!]$ for any arbitrary context $\mathcal{C}$, because e could appear in a function call when put inside $\mathcal{C}$, and that function might demand more that one value from e, because of the plurarity of $\pi^{\alpha}CRWL$. We can see this considering the program from Example 1.1 (page 3) extended with a function *coin* defined by $\{coin \to 0, coin \to 1\}$, the context $f(c([]))$ and the expression *coin*: in order to compute the value $d(0,1) \in [\![f(c(coin))]\!]$ we need $\{0,1\} \subseteq [\![coin]\!]$ while a single value of *coin* is not enough, which is reflected in the fact that $d(0,1) \in [\![f(c(0\ ?\ 1))]\!]$ while $d(0,1) \notin [\![f(c(0))]\!] \cup [\![f(c(1))]\!]$. On the other hand, note that we only need a finite subset of the denotation of the expression put in context, but not the whole denotation, which could be infinite thus leading to $t_1\ ?\ \ldots\ ?\ t_n$ being a malformed expression, as we only consider finite expressions in this work. To illustrate this we may consider again the program from Example 1.1, the symbols $z \in CS^0, s \in CS^1$ for the Peano natural numbers representation, and the function *from* defined as $\{from(X) \to X, from(X) \to s(from(X))\}$. Then, using the same context as above and the expression $from(z)$, in order to compute $d(z,s(z)) \in [\![f(c(from(z)))]\!]$ we just need $\{z,s(z)\} \subseteq [\![from(z)]\!]$, but not the infinite set of elements in $[\![from(z)]\!]$. The intuition behind this is that, as we use c-terms as values and c-terms are finite, then any computation of a value is a finite process that only involves a finite amount of information: in this case a finite subset of the denotation of the expression put in context.

Besides compositionality, $\pi^{\alpha}CRWL$ enjoys other nice properties, like the following polarity property.

Proposition 1 (Polarity of $\pi^{\alpha}CRWL$)**.** *For any program $\mathcal{P}$, $e, e' \in Exp_{\bot}$, $t, t' \in CTerm_{\bot}$ if $e \sqsubseteq e'$ and $t' \sqsubseteq t$ then $\mathcal{P} \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$ implies $\mathcal{P} \vdash_{\pi^{\alpha}CRWL} e' \twoheadrightarrow t'$ with a proof of the same size or smaller.*

Proof. By a simple induction on the structure of $e \twoheadrightarrow t$ using basic properties of $\sqsubseteq$, see Appendix A (page 38) for details. □

$\pi^\alpha CRWL$ also has some monotonicity properties related to substitutions. These are formulated using the preorder $\sqsubseteq_\pi$ over $CSubst^?_\perp$ defined by $\theta \sqsubseteq_\pi \theta'$ iff $\forall X \in \mathcal{V}$, given $\theta(X) = t_1 \ ? \ \ldots \ ? \ t_n$ and $\theta'(X) = t'_1 \ ? \ \ldots \ ? \ t'_m$ then $\forall t \in \{t_1, \ldots, t_n\} \exists t' \in \{t'_1, \ldots, t'_m\}$ such that $t \sqsubseteq t'$; and the preorder $\trianglelefteq^{\alpha pl}$ over $Subst_\perp$ defined by $\sigma \trianglelefteq^{\alpha pl} \sigma'$ iff $\forall X \in \mathcal{V}$, $[\![\sigma(X)]\!]^{\alpha pl} \subseteq [\![\sigma'(X)]\!]^{\alpha pl}$.

Proposition 2 (Monotonicity for substitutions of $\pi^\alpha CRWL$)**.** *For any program, $e \in Exp_\perp$, $t \in CTerm_\perp$, $\sigma, \sigma' \in Subst_\perp$, $\theta, \theta' \in CSubst^?_\perp$:*

1. ***Strong monotonicity of $Subst_\perp$:*** *If $\forall X \in \mathcal{V}, s \in CTerm_\perp$ given $\mathcal{P} \vdash_{\pi^\alpha CRWL} \sigma(X) \twoheadrightarrow s$ with size K we also have $\mathcal{P} \vdash_{\pi^\alpha CRWL} \sigma'(X) \twoheadrightarrow s$ with size $K' \leq K$, then $\vdash_{\pi^\alpha CRWL} e\sigma \twoheadrightarrow t$ with size L implies $\vdash_{\pi^\alpha CRWL} e\sigma' \twoheadrightarrow t$ with size $L' \leq L$.*
2. ***Monotonicity of $CSubst_\perp$:*** *If $\theta, \theta' \in CSubst_\perp$ and $\theta \sqsubseteq \theta'$ then $\mathcal{P} \vdash_{\pi^\alpha CRWL} e\theta \twoheadrightarrow t$ with size K implies $\mathcal{P} \vdash_{\pi^\alpha CRWL} e\theta' \twoheadrightarrow t$ with size $K' \leq K$.*
3. ***Monotonicity of $Subst_\perp$:*** *If $\sigma \trianglelefteq^{\alpha pl} \sigma'$ then $[\![e\sigma]\!]^{\alpha pl} \subseteq [\![e\sigma']\!]^{\alpha pl}$.*
4. ***Monotonicity of $CSubst^?_\perp$:*** *If $\theta \sqsubseteq_\pi \theta'$ then $[\![e\theta]\!]^{\alpha pl} \subseteq [\![e\theta']\!]^{\alpha pl}$.*

The properties of $\pi^\alpha CRWL$ we have seen so far are shared with $CRWL$, which is something natural taking into account that $\pi^\alpha CRWL$ is a modification of that semantics. Nevertheless, there are some properties of $CRWL$—and as a consequence, of call-time choice—that do not hold for $\pi^\alpha CRWL$. One of these is the correctness of the *bubbling* operational rule [2], which can be formulated as "under any program and for any $\mathcal{C} \in Cntxt$, $e_1, e_2 \in Exp_\perp$ we have that $[\![\mathcal{C}[e_1 \ ? \ e_2]]\!] = [\![\mathcal{C}[e_1] \ ? \ \mathcal{C}[e_2]]\!]$". Note that Examples 1.1 and 1.2 already show that this property does not hold for run-time choice, the following (counter)example proves that it is not the case for $\pi^\alpha CRWL$ neither.

Example 3.2. *Consider the program $\mathcal{P} = \{pair(X) \to (X, X), X \ ? \ Y \to X, X \ ? \ Y \to Y\}$ and the expressions $pair(0 \ ? \ 1)$ and $pair(0) \ ? \ pair(1)$ which correspond to a bubbling step using $\mathcal{C} = pair([])$. It is easy to check that $(0, 1) \in [\![pair(0 \ ? \ 1)]\!]^{\alpha pl}$ while $(0, 1) \notin [\![pair(0) \ ? \ pair(1)]\!]^{\alpha pl}$.*

It was very enlightening for us to discover that the correctness of bubbling does not hold for $\pi^\alpha CRWL$, and in fact in [45] it was wrongly considered as true. This shows that $CRWL$ and $\pi^\alpha CRWL$ are more different that it may appear at a first sight. In particular, regarding to bubbling, the important difference is that while $\pi^\alpha CRWL$ is compositional w.r.t. subsets of the denotations, $CRWL$ is compositional w.r.t. single elements of the denotation [34, 30]. Compositionality w.r.t. single values is stronger than compositionality w.r.t. subsets of the denotation, as the former implies the latter, and this also is exemplified by the fact that we need compositionality w.r.t. single values for bubbling to be correct, as we will see soon. On the other hand, compositionality w.r.t. subsets of the denotation is enough to obtain the result expressed at the end of Theorem 1, showing that expressions with the same values are indistinguishable, which corresponds to the value-based philosophy of FLP.

As the bubbling rule is devised to improve the efficiency of computations [2], it would be nice to be able to use it in some situations, although it would only be for a restricted class of contexts. In this line, we have found that bubbling is still correct under $\pi^\alpha CRWL$ for a particular kind of contexts called *constructor contexts* or just *c-contexts*, which are contexts whose holes appear under a nested application of constructor symbols only, that is, $c\mathcal{C} ::= [\,] \mid c(e_1, \ldots, c\mathcal{C}, \ldots, e_n)$, with $c \in CS^n, e_1, \ldots, e_n \in Exp_\perp$. For c-contexts, $\pi^\alpha CRWL$ enjoys the same compositionality for single values as $CRWL$—that property holds in $CRWL$ for arbitrary contexts—, as shown in the following result.

Proposition 3 (Compositionality of $\pi^\alpha CRWL$ for c-contexts)**.** *For any program, c-context $c\mathcal{C}$ and $e \in Exp_\perp$:*

$$[\![c\mathcal{C}[e]]\!]^{\alpha pl} = \bigcup_{t \in [\![e]\!]^{\alpha pl}} [\![c\mathcal{C}[t]]\!]^{\alpha pl}$$

Proof. Very similar to the proof for the general compositionality of $\pi^\alpha CRWL$ from Theorem 1, see Appendix A (page 40) for details. □

As compositionality for single values is the key property needed for bubbling to be correct, we get the following result for bubbling in $\pi^\alpha CRWL$.

Proposition 4 (Bubbling for c-contexts in $\pi^\alpha CRWL$)**.** *For any program, c-context $c\mathcal{C}$ and $e_1, e_2 \in Exp_\perp$, $[\![c\mathcal{C}[e1 \ ? \ e_2]]\!]^{\alpha pl} = [\![c\mathcal{C}[e_1] \ ? \ c\mathcal{C}[e_2]]\!]^{\alpha pl}$.*

Proof. It is easy to prove that $\forall e_1, e_2 \in Exp_\perp$ we have $[\![e1\ ?\ e_2]\!]^{\alpha pl} = [\![e_1]\!]^{\alpha pl} \cup [\![e_2]\!]^{\alpha pl}$ (see Appendix A). But then:

$$
\begin{array}{ll}
& [\![c\mathcal{C}[e1\ ?\ e_2]]\!]^{\alpha pl} \\
= & \bigcup_{t \in [\![e1\ ?\ e_2]\!]^{\alpha pl}} [\![c\mathcal{C}[t]]\!]^{\alpha pl} \qquad \text{by Proposition 3} \\
= & \bigcup_{t \in [\![e1]\!]^{\alpha pl} \cup [\![e_2]\!]^{\alpha pl}} [\![c\mathcal{C}[t]]\!]^{\alpha pl} \\
= & \bigcup_{t \in [\![e1]\!]^{\alpha pl}} [\![c\mathcal{C}[t]]\!]^{\alpha pl} \cup \bigcup_{t [\![e_2]\!]^{\alpha pl}} [\![c\mathcal{C}[t]]\!]^{\alpha pl} \\
= & [\![c\mathcal{C}[e_1]]\!]^{\alpha pl} \cup [\![c\mathcal{C}[e_2]]\!]^{\alpha pl} \qquad \text{by Proposition 3} \\
= & [\![c\mathcal{C}[e_1]\ ?\ c\mathcal{C}[e_2]]\!]^{\alpha pl}
\end{array}
$$

□

We end our presentation of $\pi^\alpha CRWL$ with an example showing how we can use $\pi^\alpha CRWL$ to model problems in which some collecting work has to be done.

Example 3.3. *We want to represent the database of a bank in which we hold some data about its employees. This bank has several branches and we want to organize the information according to them. To do that we define a non-deterministic function branches to represent the set of branches: a set is then identified with a non-deterministic expression. We also use this technique to define non-deterministic function employees which conceptually returns, for a given branch, the set of records containing the information regarding the employees that work in that branch. Now we want to search for the names of two clerks, which may be working in different branches. To do that we define the function twoclerks which is based upon the function find, which forces the desired pattern $e(N, G, clerk)$ over the set defined by the expression $employees(branches)$:*

$$
\begin{array}{rl}
\mathcal{P} = & \{branches \to madrid, \\
& \ branches \to vigo, \\
& \ employees(madrid) \to e(pepe, man, clerk), \\
& \ employees(madrid) \to e(paco, man, clerk), \\
& \ employees(vigo) \to e(maria, woman, clerk), \\
& \ employees(vigo) \to e(jaime, woman, clerk), \\
& \ twoclerks \to find(employees(branches)), \\
& \ find(e(N, G, clerk)) \to (N, N)\}
\end{array}
$$

With term rewriting $twoclerks \to find(employees(branches)) \not\to^* (pepe, maria)$, because in that expression the evaluation of branches is needed and thus one of the branches must be chosen. On the other hand with $\pi^\alpha CRWL$ the value $(pepe, maria)$ can be computed for twoclerks (some steps have been omitted for the sake of conciseness, emps abbreviates employees, and brs abbreviates branches):

$$
\dfrac{\dfrac{\dfrac{\dots}{emps(brs) \twoheadrightarrow e(pepe, \perp, clerk)}\ \textbf{POR}^\alpha \qquad \dfrac{\dots}{(pepe\ ?\ maria, pepe\ ?\ maria) \twoheadrightarrow (pepe, maria)}\ \textbf{DC}}{\dfrac{find(emps(brs)) \twoheadrightarrow (pepe, maria)}{twoclerks \twoheadrightarrow (pepe, maria)}\ \textbf{POR}^\alpha}\ \textbf{POR}^\alpha}{\dfrac{\dots}{emps(brs) \twoheadrightarrow e(maria, \perp, clerk)}\ \textbf{POR}^\alpha}\ \textbf{POR}^\alpha
$$

where

$$
\dfrac{\dfrac{}{brs \twoheadrightarrow madrid}\ \textbf{POR}^\alpha \qquad \dfrac{\dots}{e(pepe, man, clerk) \twoheadrightarrow e(pepe, \perp, clerk)}\ \textbf{DC}}{emps(brs) \twoheadrightarrow e(pepe, \perp, clerk)}\ \textbf{POR}^\alpha
$$

3.2 $\pi^\beta CRWL$

So far we have presented our first proposal for a plural semantics for constructor systems, seen some interesting properties, and how to use it to solve collecting problems. Nevertheless this semantics has also some weak points, that will be illustrated by the following example.

Example 3.4. *Starting from the program of Example 3.3, we want to search for the names of two clerks paired with their corresponding genders. Therefore, following the same ideas, we define a function find2NG that forces the desired pattern but now returning both the name and the gender of two clerks, by the rule $find2NG(e(N, G, clerk)) \to ((N, G), (N, G))$. Then, $((pepe, man), (maria, woman))$ would be one of the values computed for the expression $find2NG(employees(branches))$, as expected. Nevertheless we can also compute the value $((pepe, woman), (maria, man))$, which obviously does not correspond to the*

intended meaning of find2NG, as can be seen in the following proof (using the abbreviations above and also m for man, and w for woman).

$$\frac{\begin{array}{ll} emps(brs) \twoheadrightarrow e(pepe, m, clerk) & ((pepe\ ?\ maria, m\ ?\ w), (pepe\ ?\ maria, m\ ?\ w)) \\ emps(brs) \twoheadrightarrow e(maria, w, clerk) & \quad \twoheadrightarrow ((pepe, w), (maria, m)) \end{array}}{find2NG(emps(brs)) \twoheadrightarrow ((pepe, w), (maria, m))}\ \boldsymbol{POR}^{\alpha}$$

This example is interesting because it shows a relevant flaw of $\pi^{\alpha}CRWL$, since there the matching substitutions $[N/pepe, G/man]$ and $[N/maria, G/woman]$ obtained for the different evaluations of the argument $employees(branches)$ become wrongly intermingled. Anyway the program is not well conceived, as it does not specify that each of the (N, G) pairs correspond to a particular clerk in the database, thus preventing an unintended information mixup. Nevertheless a better semantic behaviour would have prevented "mixed" results like $((pepe, woman), (maria, man))$ thus getting $((maria, woman), (maria, woman))$ and $((pepe, man), (pepe, man))$ as the only total values for $find2NG(employees(branches))$, which does not fix the program but at least avoids wrong information mixup.[6]

This problem was also pointed out in [9], where an identification between $d(0,0)\ ?\ d(1,1)$ and $d(0\ ?\ 1, 0\ ?\ 1)$—for $d \in CS^2$ and $0, 1 \in CS^0$—made by $\pi^{\alpha}CRWL$ for relevant contexts was reported. In the technical setting presented in that paper another plural semantics that avoids this problem is proposed, although its technical relation with call-time or run-time choice is not formally stated nor proved. In that work, that particular plurality is achieved by allowing bubbling steps for constructor applications by means of a rule that could be expressed in our syntax as $[\![c(e_1, \ldots, e'_1\ ?\ e'_2, \ldots, e_n)]\!] = [\![c(e_1, \ldots, e'_1, \ldots, e_n)\ ?\ c(e_1, \ldots, e'_2, \ldots, e_n)]\!]$. This kind of rules are well suited for a step-wise semantics like the one presented in [9], but are more difficult to integrate with a goal-oriented proof calculus in the style of *CRWL* or $\pi^{\alpha}CRWL$, which—as we saw in the presentation of $\pi^{\alpha}CRWL$ above—perform a kind of innermost evaluation of expressions by exploiting the use of partial values to get a compositional calculus for a lazy semantics.

Hence, in order to adapt this idea to our framework, we could *switch from bubbling under constructors to bubbling of* $CSubst^{?}_{\perp}$*, allowing the combination of substitutions that only differ in the value they assign to a single variable.* We could then for example perform the following bubbling derivation of substitutions.

$$\begin{array}{l} \underline{[X/0, Y/0] \sqcup [X/0, Y/1]} \sqcup [X/1, Y/0] \sqcup [X/1, Y/1] \\ \rightarrow_{\sqcup} [X/0, Y/0\ ?\ 1] \sqcup \underline{[X/1, Y/0]} \sqcup [X/1, Y/1] \\ \rightarrow_{\sqcup} \underline{[X/0, Y/0\ ?\ 1] \sqcup [X/1, Y/0\ ?\ 1]} \rightarrow_{\sqcup} [X/0\ ?\ 1, Y/0\ ?\ 1] \end{array}$$

This derivation shows a criterion that determines that the set of c-substitutions $\{[X/0, Y/0], [X/0, Y/1], [X/1, Y/0], [X/1, Y$ can be safely combined into $[X/0\ ?\ 1, Y/0\ ?\ 1] \in CSubst^{?}_{\perp}$ with no wrong substitution mixup. On the other hand, for $[X/0, Y/0] \sqcup [X/1, Y/1]$ we should not be able to perform any $\rightarrow_{\sqcup}$ step as these substitutions differ in more than one variable, thus failing to combine those c-substitutions into a single element from $CSubst^{?}_{\perp}$. As can be seen in Figure 2, the key for getting a plural behaviour in $\pi^{\alpha}CRWL$ is finding a way to combine different matching substitutions obtained from the evaluation of the same expression, therefore this new combination method should give rise to another plural semantic proposal. We conjecture that the resulting semantics expresses the same plural semantics proposed in [9]—the one resulting in that setting when only variables of sort *Ch* (as defined in that paper) are used—, although we will not give any formal result relating both proposals. Let us call $\pi^{\beta}CRWL$ to this new semantics in which parameter passing is only perfomed with substituions from $CSubst^{?}_{\perp}$ that come from a succesful combinations of c-substitutions using the relation $\rightarrow_{\sqcup}$, and consider the behaviour of the different plural semantics in the following example.

Example 3.5. *Consider the constructors* $c \in CS^1$, $d \in CS^2$, $l \in CS^4$ *and* $0, 1 \in CS^0$, *and the following program.*

$$\begin{array}{ll} f(c(X)) \rightarrow d(X, X) & h(d(X, Y)) \rightarrow d(X, X) \\ g(d(X, Y)) \rightarrow l(X, X, Y, Y) & k(d(X, Y)) \rightarrow d(X, Y) \end{array}$$

- $f(c(0)\ ?\ c(1))$ *and* $f(c(0\ ?\ 1))$ *behave the same in both* $\pi^{\alpha}CRWL$ *and* $\pi^{\beta}CRWL$, *because there is only one variable involved in the matching substitution and thus no substitution mixup like the ones seen before may appear, as for both expressions we only have to combine the substitutions* $[X/0]$ *and* $[X/1]$. *We can reach the values* $d(0,0)$, $d(0,1)$, $d(1,0)$, *and* $d(1,1)$ *for each expression in both semantics.*

[6] In Section 5 we will see how to combine singular and plural *function arguments* to solve a generalization of this problem.

- *More surprisingly we also get the same behaviour for $h(d(0,0) \ ? \ d(1,1))$ and $h(d(0 \ ? \ 1, 0 \ ? \ 1))$ in both π^{α}CRWL and π^{β}CRWL. There the suspicious expression is $h(d(0,0) \ ? \ d(1,1))$ which generates the matching substitutions $[X/0, Y/0]$ and $[X/1, Y/1]$ which are wrongly combined by π^{α}CRWL into the substitution $?\{[X/0, Y/0], [X/1, Y/1]\} = [X/0 \ ? \ 1, Y/0 \ ? \ 1]$, used to instantiate the right-hand side of the rule for h. But this mistake has no consequence because only X appears in the right-hand side of the rule for h, therefore it has the same effect as combining $[X/0, Y/\ \bot]$ and $[X/1, Y/\ \bot]$ into $[X/0 \ ? \ 1, Y/\ \bot]$, which is just what is done in π^{β}CRWL as we will see later on.*

  On the other hand $h(d(0 \ ? \ 1, 0 \ ? \ 1))$ is not problematic as it generates the matching substitutions $[X/0, Y/0]$, $[X/0, Y/1]$, $[X/1, Y/0]$ and $[X/1, Y/1]$ that already cover all the possible instantiations of X and Y caused by its combination in π^{α}CRWL, the substitution $[X/0 \ ? \ 1, Y/0 \ ? \ 1]$. The point is that in a sense both $\{[X/0, Y/0], [X/0, Y/1], [X/1, Y/0], [X/1, Y/1]\}$ and $[X/0 \ ? \ 1, Y/0 \ ? \ 1]$ have the same power. This will also be reflected by the fact that π^{β}CRWL would be able to combine the former set into the latter $CSubst^{?}_{\bot}$.

  Again, we can reach the values $d(0,0)$, $d(0,1)$, $d(1,0)$ and $d(1,1)$ for each expression in both semantics.

- *It is for the expressions $g(d(0,0) \ ? \ d(1,1))$ and $g(d(0 \ ? \ 1, 0 \ ? \ 1))$ that we can see a different behaviour of π^{α}CRWL and π^{β}CRWL. Once again $g(d(0 \ ? \ 1, 0 \ ? \ 1))$ is not problematic, and for it we can get the values $l(0,0,0,0)$, $l(0,0,0,1)$, ... and all the combinations of 0 and 1, in both semantics. But for $g(d(0,0) \ ? \ d(1,1))$ we have that, for example, to compute $l(0,0,0,1)$ we need the expression $d(0,0) \ ? \ d(1,1)$ to generate both 0 and 1 for Y in the matching substitutions. The only (total) matching substitutions that can be obtained from the evaluation of $d(0,0) \ ? \ d(1,1)$ are $[X/0, Y/0]$ and $[X/1, Y/1]$, which cannot be combined by π^{β}CRWL, hence we cannot get both 0 and 1 for Y in the combined substitution. As a consequence $l(0,0,0,0)$ and $l(1,1,1,1)$ are the only values computed for $g(d(0,0) \ ? \ d(1,1))$ by π^{β}CRWL. On the other hand, π^{α}CRWL computes all the combinations of 0 and 1—like it did for $g(d(0 \ ? \ 1, 0 \ ? \ 1))$—, as it is able to combine $\{[X/0, Y/0], [X/1, Y/1]\}$ into $[X/0 \ ? \ 1, Y/0 \ ? \ 1]$.*

- *A more exotic discovery is that $k(d(0,0) \ ? \ d(1,1))$ does not behave the same for call-time choice, run-time choice, π^{α}CRWL, and π^{β}CRWL, even though it only uses a right-linear program rule, and it is a known fact that call-time choice and run-time choice are equivalent for right-linear programs [28]. CRWL (call-time choice), term rewriting (run-time choice), and π^{β}CRWL only compute the values $d(0,0)$ and $d(1,1)$ for $k(d(0,0) \ ? \ d(1,1))$, in the case of π^{β}CRWL because it fails to combine $[X/0, Y/0]$ and $[X/1, Y/1]$. Nevertheless π^{α}CRWL is able to combine those substitutions into $[X/0 \ ? \ 1, \ Y/0 \ ? \ 1]$, thus getting the additional values $d(0,1)$ and $d(1,0)$ for the expression $k(d(0,0) \ ? \ d(1,1))$. However we still strongly conjecture that π^{β}CRWL—as formulated below—is equivalent to call-time and run-time choice for right-linear programs.*

The previous example motivates the interest of a formal definition of π^{β}CRWL. It would be nice if it were by means of a proof calculus similar to CRWL and π^{α}CRWL, because then their comparison would be easier, and maybe they could even share some of their properties, in particular compositionality. The ideas above regarding bubbling derivations for substitutions have given us the right intuitions, but those derivations are not so easy to handle as the following characterization of *compressible sets of c-substitutions*, which will be the only sets of substitutions that will be combined by π^{β}CRWL.

Definition 1 (Compressible set of $CSubst_{\bot}$)**.**
A finite set $\Theta \subseteq CSubst_{\bot}$ is compressible *iff for $\{X_1, \ldots, X_n\} = \bigcup_{\theta \in \Theta} dom(\theta)$*

$$\{(X_1\theta, \ldots, X_n\theta) \mid \theta \in \Theta\} = \{X_1\theta_1 \mid \theta_1 \in \Theta\} \times \ldots \times \{X_n\theta_n \mid \theta_n \in \Theta\}$$

Note that this property is easily computable for Θ finite, as we only consider finite domain substitutions.

Example 3.6. *Let us see how the notion of compressible set of c-substitutions can be used to replace the relation $\rightarrow_{\sqcup}$ sketched above. We have seen that the substituions $[X/0, Y/0]$ and $[X/1, Y/1]$ should not be combined in order to prevent a wrong substitution mixup. This is reflected in the fact that the set $\{[X/0, Y/0], [X/1, Y/1]\}$ is not compressible, because:*

$$\begin{array}{l} \{(X\theta, Y\theta) \mid \theta \in \{[X/0, Y/0], [X/1, Y/1]\}\} = \{(0,0), (1,1)\} \\ \neq \{(0,0), (0,1), (1,0), (1,1)\} \\ = \{0,1\} \times \{0,1\} = \{X\theta_x \mid \theta_x \in \Theta\} \times \{Y\theta_y \mid \theta_y \in \Theta\} \end{array}$$

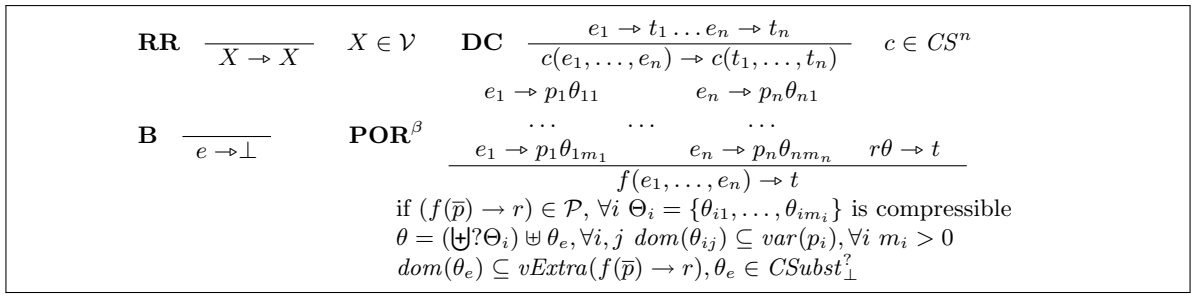

Figure 3: Rules of $\pi^\beta CRWL$

On the other hand for $\Theta = \{[X/0,Y/0],[X/0,Y/1],[X/1,Y/0],[X/1,Y/1]\}$ *the substitutions it contains can be safely combined, therefore we should have that* Θ *is compressible, as it happens:*

$$\{(X\theta, Y\theta) \mid \theta \in \Theta\} = \{(0,0),(0,1),(1,0),(1,1)\}$$
$$= \{0,1\} \times \{0,1\} = \{X\theta_x \mid \theta_x \in \Theta\} \times \{Y\theta_y \mid \theta_y \in \Theta\}$$

Our last proposal for a plural semantics for CS's is based on the notion of compressible set of c-substitutions, and it is defined by the $\pi^\beta CRWL$-proof calculus in Figure 3. Note that the only difference with $\pi^\alpha CRWL$ is that the rule $\mathbf{POR}^\alpha$ is replaced by $\mathbf{POR}^\beta$, that now demands the different matching substitutions obtained from the evaluation of each function argument to be compressible. Apart from that, compressible sets of partial c-substitutions are combined just like in $\pi^\alpha CRWL$, by means of the ? operator.

This calculus, like *CRWL* and $\pi^\alpha CRWL$, also derives *reduction statements* of the form $\mathcal{P} \vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$, which expresses that t is (or approximates to) a possible value for e in this semantics, under the program $\mathcal{P}$. Then the $\pi^\beta CRWL$*-denotation* of an expression $e \in Exp_\bot$ under a program $\mathcal{P}$ in $\pi^\beta CRWL$ is defined as $[\![e]\!]^{\beta pl}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{\pi^\beta CRWL} e \twoheadrightarrow t\}$. In the following, we will usually omit the reference to $\mathcal{P}$ when implied by the context.

Example 3.7. *Consider the program of Example 3.4,* $\pi^\beta CRWL$ *is able to avoid computing the value* $((pepe, woman),(maria, man))$ *for the expression* $find2NG(employees(branches))$ *because the set of matching substitutions* $\{[N/pepe, G/man],[N/maria, G/woman]\}$ *is not compressible, as can be easily checked by applying Definition 1 in a way similar to Example 3.6. Nevertheless, the values* $((maria, woman),(maria, woman))$ *and* $((pepe, man),(pepe, man))$ *can be computed for* $find2NG(employees(branches))$ *by using the sets of substitutions* $\{[N/pepe, G/man]\}$ *and* $\{[N/maria, G/woman]\}$*, respectively, for parameter passing, which are compressible as they are singleton. As we saw in Example 3.4, the function find2NG is wrongly conceived because it does not specify that the in each pair* (N, G) *the name* N *and the genre* G *must correspond to the same clerk.* $\pi^\beta CRWL$ *cannot fix a wrong program, but at least is able to prevent "mixed" results like* $((pepe, woman),(maria, man))$*.*

It is also easy to check that $\pi^\beta CRWL$ *has the same behaviour that* $\pi^\alpha CRWL$ *for Example 3.3, as sets like* $\{[N/pepe, G/\bot],[N/maria, G/\bot]\}$ *are compressible. Similarly, in Example 3.5 the functions* f *and* h *behave the same under both semantics, and* $\pi^\beta CRWL$ *also behaves for* h *and* k *as specified there, because* $\{[X/0,Y/0],[X/1,Y/1]\}$ *is not compressible, just like* $\{[N/pepe, G/man],[N/maria, G/woman]\}$*, while for* $\Theta = \{[X/0,Y/0],[X/0,Y/1],[X/1,Y/0],[X/1,Y/1]\}$ *we have that* Θ *is compressible, as seen in Example 3.6.*

The following result shows that part of the equality that defines compressibility always holds trivially, thus simplifying the definition of compressible set of c-substitutions.

Lemma 2. *For any finite set* $\Theta \subseteq CSubst_\bot$ *for* $\{X_1,\ldots,X_n\} = \bigcup_{\theta\in\Theta} dom(\theta)$ *we have*

$$\{(X_1\theta,\ldots,X_n\theta) \mid \theta \in \Theta\} \subseteq \{X_1\theta_1 \mid \theta_1 \in \Theta\} \times \ldots \times \{X_n\theta_n \mid \theta_n \in \Theta\}$$

As a consequence Θ *is compressible iff*

$$\{(X_1\theta,\ldots,X_n\theta) \mid \theta \in \Theta\} \supseteq \{X_1\theta_1 \mid \theta_1 \in \Theta\} \times \ldots \times \{X_n\theta_n \mid \theta_n \in \Theta\}$$

This gives another criterion to prove compressibility: Θ *is compressible iff* $\forall\theta_1,\ldots,\theta_n \in \Theta.\ \exists\theta \in \Theta$ *such that* $\forall i. X_i\theta_i \equiv X_i\theta$ *(which implies that* $(X_1\theta_1,\ldots,X_n\theta_n) \equiv (X_1\theta,\ldots,X_n\theta)$*).*

In a way this result exemplifies why $\pi^\beta CRWL$ is smaller than $\pi^\alpha CRWL$ in the sense that in general it computes less values for a given expression under a given program, as $\{X_1\theta_1 \mid \theta_1 \in \Theta\} \times \ldots \times \{X_n\theta_n \mid \theta_n \in \Theta\}$ corresponds to the substitution $?\Theta$ that is always used for parameter passing in $\pi^\alpha CRWL$, with no previous compressibility test. We will see more about the relations between call-time choice, run-time choice, $\pi^\alpha CRWL$, and $\pi^\beta CRWL$ in Section 4.

We have just seen how $\pi^\beta CRWL$ corrects the excessive permissiveness of the combinations of substitutions performed by $\pi^\alpha CRWL$ but, will it be able to do it while keeping the nice properties of $\pi^\alpha CRWL$—in particular compositionality—at the same time? Fortunately the answer is yes, as shown by the following result.

Theorem 2 (Basic properties of $\pi^\beta CRWL$)**.** *The basic properties of $\pi^\alpha CRWL$ also hold for $\pi^\beta CRWL$ under any program, i.e, the corresponding versions of Theorem 1, Proposition 1, Proposition 2, Proposition 3, and Proposition 4 also hold for $\pi^\beta CRWL$.*

For Proposition 2 in particular we replace $\trianglelefteq^{\alpha pl}$ with $\trianglelefteq^{\beta pl}$, which is defined in terms of $\pi^\beta CRWL$ instead of $\pi^\alpha CRWL$. Nevertheless, in the following we will often omit the superscripts αpl and βpl in $\trianglelefteq^{\alpha pl}$ and $\trianglelefteq^{\beta pl}$ when those are implied by the context.

Proof. In each proof for the $\pi^\alpha CRWL$ versions of these results we start from a given $\pi^\alpha CRWL$-proof and build another one using a bigger expression w.r.t. $\sqsubseteq$, a more powerful substitution, interchanging an expression with an alternative of some of its values... Therefore we can use the same technique for $\pi^\beta CRWL$ to replicate any $\mathbf{POR}^\beta$ step in the starting $\pi^\beta CRWL$-proof by using the substitution used there for parameter passing, which must be compressible by hypothesis, and that we are able to obtain by using a similar reasoning to that performed in the proof for the corresponding result for $\pi^\alpha CRWL$. □

In this section we have presented two different proposals for a plural semantics for non-deterministic constructor systems that are different from run-time choice. The first one, $\pi^\alpha CRWL$, is a pretty simple extension of *CRWL* that comes up naturally from allowing the combination of several matching substitution through the operator ? for c-substitutions. But it is precisely the simplicity of that combination which leads to a wrong information mixup in some situations. These problems are solved in $\pi^\beta CRWL$, in which a compressibility test is added to prevent a wrong combinations of substitutions. This could suggest that $\pi^\alpha CRWL$ is only a preliminary attempt that should now be put aside and forgotten. Nevertheless $\pi^\alpha CRWL$ will still be very useful for us, again because of its simplicity, as we will see in subsequent sections.

Finally note that both $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ have been devised starting from *CRWL* and then adding some criterion for combining different matching substitutions for the same argument, so any number of alternative plural—and even also compositional, possibly—semantics for constructor systems could be conceived just by defining new combination procedures.

4 Hierarchy, equivalence, and simulation

In this section we will first compare the different characteristics of the semantics considered so far, with a special emphasis in the set of computed c-terms. Then we will present a class of programs characterized by a simple syntactic criterion under which our two plural semantics are equivalent. Finally we will conclude the section presenting a program transformation that can be used to simulate our plural semantics by using term rewriting.

4.1 A hierarchy of semantics

We have already seen that *CRWL*, $\pi^\alpha CRWL$, and $\pi^\beta CRWL$ enjoy similar properties like polarity, monotonicity for substitutions and, above all, compositionality, which implies that two expressions have the same denotation if and only if they have the same denotation when put under the same arbitrary context. This is not the case for run-time choice, as we saw when switching from $f(c(0\ ?\ 1))$ to $f(c(0)\ ?\ c(1))$ in Examples 1.1 and 1.2, taking into account that for the expressions $c(0\ ?\ 1)$ and $c(0)\ ?\ c(1)$ the same values are computed under run-time choice, i.e., the same c-terms are reached by a term rewriting derivation.[7]

[7] In fact compositionality can be achieved for run-time choice by using a different set of values instead of the partial c-terms considered in this work. Those values essentially are recursively nested applications of constructor symbols to sets of values structured in the same way, therefore intrinsically more complicated that plain c-terms, and anyway not considered in the present work—See [31] for details.

But our main goal in this section is to study the relationship between call-time choice, run-time choice, $\pi^\alpha CRWL$, and $\pi^\beta CRWL$ w.r.t. the denotations they define, which express the set of values computed by each semantics. To do that we will lean on a traditional notion from the *CRWL* framework, the notion of *shell* $|e|$ *of an expression* e, which represents the outer constructor (thus partially computed) part of e, defined as $|\bot| = \bot$, $|X| = X$, $c(e_1, \ldots, e_n) = c(|e_1|, \ldots, |e_n|)$, $|f(e_1, \ldots, e_n)| = \bot$, for $X \in \mathcal{V}, c \in CS, f \in FS$. Now we can define our notion of denotation of an expression in each of the semantics considered.

Definition 2 (Denotations)**.** *For any program* $\mathcal{P}$*,* $e \in Exp$ *we define the denotation of* e *under the different semantics as follows*

- $[\![e]\!]^{sg}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t\}$.
- $[\![e]\!]^{rt}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash e \rightarrow^* e' \wedge t \sqsubseteq |e'|\}$.
- $[\![e]\!]^{\alpha pl}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t\}$.
- $[\![e]\!]^{\beta pl}_{\mathcal{P}} = \{t \in CTerm_\bot \mid \mathcal{P} \vdash_{\pi^\beta CRWL} e \twoheadrightarrow t\}$.

In the following, we will usually omit the reference to $\mathcal{P}$ *when implied by the context.*

As $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ are modifications of *CRWL*, the relation between these three semantics is straightforward.

Theorem 3. *For any CRWL-program* $\mathcal{P}$*,* $e \in Exp_\bot$

$$[\![e]\!]^{sg} \subseteq [\![e]\!]^{\beta pl} \subseteq [\![e]\!]^{\alpha pl}$$

None of the converse inclusions hold in general.

Proof. Given a *CRWL*-proof for $\vdash_{CRWL} e \twoheadrightarrow t$ we can build a $\pi^\alpha CRWL$-proof for $\vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ just replacing every **OR** step by the corresponding $\mathbf{POR}^\beta$ step, as it is easy to see that any singleton set of c-substitutions is compressible, and that $?\{\theta\} = \theta$. As a consequence $[\![e]\!]^{sg} \subseteq [\![e]\!]^{\beta pl}$. On the other hand we can turn any $\pi^\beta CRWL$-proof into a $\pi^\alpha CRWL$-proof just replacing any $\mathbf{POR}^\beta$ step by the corresponding $\mathbf{POR}^\alpha$, as $\mathbf{POR}^\beta$ has stronger premises than $\mathbf{POR}^\alpha$, and the same consequence. Therefore $[\![e]\!]^{\beta pl} \subseteq [\![e]\!]^{\alpha pl}$.

Regarding the failure of the converse inclusions in the general case, consider the program $\{pair(X) \rightarrow d(X,X), g(d(X,Y)) \rightarrow d(X,Y)\}$ for which it is easy to check that $[\![pair(0?1)]\!]^{sg} \not\ni d(0,1) \in [\![pair(0?1)]\!]^{\beta pl}$ and $[\![g(d(0,0)?d(1,1))]\!]^{\beta pl} \not\ni d(0,1) \in [\![g(d(0,0)?d(1,1))]\!]^{\alpha pl}$. □

Concerning the relation between call-time choice and run-time choice, it was already explored in previous works of the authors [33, 30], and we recast it here in the following theorem.

Theorem 4. *For any CRWL-program* $\mathcal{P}$*,* $e \in Exp$*,* $[\![e]\!]^{sg} \subseteq [\![e]\!]^{rt}$*. The converse inclusion does not hold in general (as shown by Example 1.1).*

On the other hand we cannot rely on any precedent in order to study the relation between $\pi^\beta CRWL$ and run-time choice. Therefore, putting run-time choice in the right place in the semantics inclusion chain from Theorem 3 will be one of the contributions of this work. We anticipate that the conclusion is that $\pi^\beta CRWL$ computes more values in general.

Theorem 5. *For any CRWL-program* $\mathcal{P}$*,* $e \in Exp$*,* $[\![e]\!]^{rt} \subseteq [\![e]\!]^{\beta pl}$*. The converse inclusion does not hold in general.*

It is easy to prove the last statement of Theorem 5, as in fact Example 1.2 is a valid counterexample for that, but proving the first part is far more complicated. The key for this proof is the following lemma stating that every term rewriting step is sound w.r.t. $\pi^\beta CRWL$.

Lemma 3 (One step soundness of $\rightarrow$ w.r.t. $\pi^\beta CRWL$)**.** *For any CRWL-program* $\mathcal{P}$*,* $e, e' \in Exp$ *if* $e \rightarrow e'$ *then* $[\![e']\!]^{\beta pl} \subseteq [\![e]\!]^{\beta pl}$.

Note that any term rewriting step is of the shape $\mathcal{C}[f(\overline{p})\sigma] \rightarrow \mathcal{C}[r\sigma]$ for some $\sigma \in Subst$ and some program rule $f(\overline{p}) \rightarrow r$. If we could prove Lemma 3 for any step performed at the root of the starting expression, i.e. that $f(\overline{p})\sigma \rightarrow r\sigma$ implies $[\![r\sigma]\!]^{\beta pl} \subseteq [\![f(\overline{p})\sigma]\!]^{\beta pl}$, then we could use the compositionality of $\pi^\beta CRWL$ from Theorem 2 to propagate the result $[\![r\sigma]\!]^{\beta pl} \subseteq [\![f(\overline{p})\sigma]\!]^{\beta pl}$ to $[\![\mathcal{C}[r\sigma]]\!]^{\beta pl} \subseteq [\![\mathcal{C}[f(\overline{p})\sigma]]\!]^{\beta pl}$. To do that we will use the following notion of $\pi^\beta CRWL$-denotation of a substitution.

Definition 3 (Denotation of substitutions). *For any CRWL-program $\mathcal{P}$, $\sigma \in Subst_\perp$ the π^βCRWL-denotation of σ under $\mathcal{P}$ is*

$$[\![\sigma]\!]^{\beta pl}_{\mathcal{P}} = \{\theta \in CSubst_\perp \mid \forall X \in \mathcal{V},\ \mathcal{P} \vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow \theta(X)\}$$

Denotations of substitutions enjoy several interesting properties, and for example every $\sigma \in Subst_\perp$ is more powerful than any combination of substitutions from its denotation, by means of the ? operator. This is something natural, because c-substitutions in $[\![\sigma]\!]^{\beta pl}$ only contain a finite part of the possibly infinite set of values generated for each expression in the range of σ.

Lemma 4. *For any finite not empty $\Theta \subseteq [\![\sigma]\!]^{\beta pl}$ we have $?\Theta \trianglelefteq^{\beta pl} \sigma$.*

Besides, it is clear that in any π^βCRWL-proof that uses some $\sigma \in Subst_\perp$ only a finite amount of the information contained in σ, and therefore in $[\![\sigma]\!]^{\beta pl}$, is employed, just like in any proof for a statement $\vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$ only a finite amount of the information in e is used, because t is a finite element and the π^βCRWL-proof is also finite, otherwise the statement $\vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$ could not have been proved. These intuitions are formalized in the following result.

Lemma 5. *For any $\sigma \in Subst_\perp, e \in Exp_\perp, t \in CTerm_\perp$ if $\vdash_{\pi^\beta CRWL} e\sigma \twoheadrightarrow t$ then $\exists \Theta \subseteq [\![\sigma]\!]^{\beta pl}$ finite and not empty such that $\vdash_{\pi^\beta CRWL} e(?\Theta) \twoheadrightarrow t$*

Proof (sketch). First we prove the case where $e \equiv X \in \mathcal{V}$. If $X \in dom(\sigma)$ then we define some $\theta \in CSubst_\perp$ as

$$\theta(Y) = \begin{cases} t & \text{if } Y \equiv X \\ \perp & \text{if } Y \in (dom(\sigma) \setminus \{X\}) \\ Y & \text{if } Y \notin dom(\sigma) \end{cases}$$

Otherwise if $X \notin dom(\sigma)$ then given $\overline{Y} = dom(\sigma)$ we define $\theta = [\overline{Y/\perp}]$. In both cases it is easy to see that taking $\Theta = \{\theta\}$ then the conditions of the lemma are granted. To prove the general case where e is not restricted to be a variable we perform an easy induction over the structure of $e\sigma \twoheadrightarrow t$, using the property that for any $\Theta, \Theta' \subseteq CSubst_\perp$, if $\Theta \subseteq \Theta'$ then $?\Theta \sqsubseteq_\pi ?\Theta'$, combined with the monotonicity under substitutions of π^βCRWL. See Appendix A (page 42) for details. □

This result is very interesting because it expresses a particular property of our plural semantics, as it can be also proved true for the corresponding definition of π^αCRWL-denotation of a substitution. The key in this result is that the substitution obtained for rebuilding the starting derivation is a substitution from $CSubst^?_\perp$, which are precisely the kind of substitutions used for parameter passing in our plural semantics. On the other hand this is not true for CRWL, and it is one of the reasons why in general call-time choice computes less values than run-time choice: just consider the derivation $\vdash_{CRWL} d(X,X)[X/0\ ?\ 1] \twoheadrightarrow d(0,1)$ for which there is no substitution θ in $CSubst_\perp$—the kind of substitutions used for parameter passing in CRWL—such that $\vdash_{CRWL} d(X,X)\theta \twoheadrightarrow d(0,1)$. Nevertheless if we restrict to deterministic programs this property becomes true for CRWL—and besides in that case run-time choice and call-time choice are equivalent too, see [33, 30] for details.

Although Lemma 5 is a nice result we still need an extra ingredient to be able to use it for proving Lemma 3, thus enabling an easy proof for Theorem 5. The point is that we cannot use an arbitrary substitution from $CSubst^?_\perp$ for parameter passing in π^βCRWL but only a substitution which would be also compressible, in order to ensure that no wrong substitution mixup is performed, which is precisely the main feature of π^βCRWL. Therefore although a version of Lemma 5 for π^αCRWL can be used for proving that term rewriting is sound w.r.t. π^αCRWL—as in fact it was done in [45]—, for proving its soundness w.r.t. π^βCRWL we will still need to do a little extra effort. And the missing piece is the following notion of compressible completion of a set of c-substitutions, which adds some additional c-substitutions to its input set in order to ensure that the resulting set is then compressible.

Definition 4 (Compressible completion). *Given $\Theta \subseteq CSubst_\perp$ finite such that $\{X_1, \ldots, X_n\} = \bigcup_{\theta \in \Theta} dom(\theta)$, its compressible completion $cc(\Theta)$ is defined as*

$$cc(\Theta) = \{[X_1/X_1\theta_1, \ldots, X_n/X_n\theta_n] \mid \theta_1, \ldots, \theta_n \in \Theta\}$$

Every compressible completion enjoys the following basic properties, which explain why we call it "completion" and also "compressible."

Proposition 5 (Properties of $cc(\Theta)$). *For any $\Theta \subseteq CSubst_\perp$ finite such that $\{X_1, \ldots, X_n\} = \bigcup_{\theta \in \Theta} dom(\theta)$*

a) $cc(\Theta) \subseteq CSubst_\bot$ *and it is finite.*

b) $\Theta \subseteq cc(\Theta)$*. As a result,* $?\Theta \sqsubseteq_\pi ?cc(\Theta)$*.*

c) $\bigcup_{\mu \in cc(\Theta)} dom(\mu) = \{X_1, \ldots, X_n\}$*.*

d) $cc(\Theta)$ *is compressible.*

But, for the current task, the most interesting property of compressible completions is the following.

Lemma 6. *For any* $\sigma \in Subst_\bot$ *and any* $\Theta \subseteq [\![\sigma]\!]^{\beta pl}$ *finite and not empty we have that* $cc(\Theta) \subseteq [\![\sigma]\!]^{\beta pl}$ *too.*

This is precisely the result we need to strengthen Lemma 5 so it now becomes applicable for $\pi^\beta CRWL$, as it allows us to shift from any subset of the $\pi^\beta CRWL$-denotation of a substitution to its compressible completion, which will be also more powerful than the starting subset thanks to Proposition 5 b).

Lemma 7. *For any* $\sigma \in Subst_\bot, e \in Exp_\bot, t \in CTerm_\bot$ *if* $\vdash_{\pi^\beta CRWL} e\sigma \twoheadrightarrow t$ *then* $\exists \Theta \subseteq [\![\sigma]\!]^{\beta pl}$ *finite, not empty, and compressible such that* $\vdash_{\pi^\beta CRWL} e(?\Theta) \twoheadrightarrow t$*.*

Proof. By Lemma 5 we get some $\Theta \subseteq [\![\sigma]\!]^{\beta pl}$ finite and not empty such that $\vdash_{\pi^\beta CRWL} e(?\Theta) \twoheadrightarrow t$. Then by Lemma 6 we get that $cc(\Theta) \subseteq [\![\sigma]\!]^{\beta pl}$ too, and that it is finite, not empty (as $\Theta \subseteq cc(\Theta)$ and Θ is not empty), compressible and $?\Theta \sqsubseteq_\pi ?cc(\Theta)$ by Proposition 5. But then we can apply the monotonicity of Theorem 2 to get $\vdash_{\pi^\beta CRWL} e(?cc(\Theta)) \twoheadrightarrow t$, so we are done. □

We can now use this result to prove a particularization of Lemma 3 (one step soundness of $\to$ w.r.t. $\pi^\beta CRWL$) for steps performed at the root of the expression, i.e., of the shape $f(\overline{p})\sigma \to r\sigma$. Thus, given some $t \in [\![r\sigma]\!]^{\beta pl}$ our goal is proving that $t \in [\![f(\overline{p})\sigma]\!]^{\beta pl}$. First of all by Lemma 7 we get some compressible $\Theta \subseteq [\![\sigma]\!]^{\beta pl}$ such that $t \in [\![r(?\Theta)]\!]^{\beta pl}$. If we could use it to prove that $t \in [\![f(\overline{p})(?\Theta)]\!]^{\beta pl}$ then by Lemma 4 we would get $?\Theta \trianglelefteq \sigma$, so by the monotonicity of Theorem 2 we could obtain $t \in [\![f(\overline{p})\sigma]\!]^{\beta pl}$ as we wanted. As $\overline{p} \subseteq CTerm_\bot$ and $\Theta \subseteq CSubst_\bot$ we can easily prove that $\forall p_i \in \overline{p}, \theta_j \in \Theta$ we have $\vdash_{\pi^\beta CRWL} p_i(?\Theta) \twoheadrightarrow p_i\theta_j$. All this can be used to perform the following step, assuming $\Theta = \{\theta_1, \ldots, \theta_m\}$.

$$\frac{\begin{array}{ccccc} p_1(?\Theta) \twoheadrightarrow p_1\theta_1 \equiv p_1\theta_1|_{var(p_1)} & & & & p_n(?\Theta) \twoheadrightarrow p_n\theta_1 \equiv p_n\theta_1|_{var(p_n)} \\ \ldots & & \ldots & & \ldots \\ p_1(?\Theta) \twoheadrightarrow p_1\theta_m \equiv p_1\theta_m|_{var(p_1)} & \ldots & & p_n(?\Theta) \twoheadrightarrow p_n\theta_m \equiv p_n\theta_m|_{var(p_n)} & r\theta' \equiv r(?\Theta) \twoheadrightarrow t \end{array}}{f(p_1, \ldots, p_n)(?\Theta) \twoheadrightarrow t}\ \mathbf{POR}^\beta$$

for $\theta' = (\biguplus ?\Theta_i) \uplus \theta_e$ where $\forall i \in \{1, \ldots, n\}.\Theta_i = \{\theta_j|_{var(p_i)} \mid \theta_j \in \Theta\}$, $\theta_e = (?\Theta)|_{\mathcal{V}_e}$ for $\mathcal{V}_e = vExtra(f(\overline{p}) \to r)$. It can be easily proved that having Θ compressible implies that each Θ_i is also compressible—so the $\mathbf{POR}^\beta$ step above is valid—, and that $r\theta' \equiv r(?\Theta)$.

Therefore we have just proved the soundness w.r.t. $\pi^\beta CRWL$ of term rewriting steps performed at the root of the starting expression. So all that is left is using the compositionality of $\pi^\beta CRWL$ from Theorem 2 for propagating this result for steps performed in an arbitrary context. A fully detailed proof for Lemma 3 can be found in Appendix A (page 44).

And now we are finally ready to prove Theorem 5.

For Theorem 5. Given some $t \in [\![e]\!]^{rt}$, by definition $\exists e' \in Exp$ such that $t \sqsubseteq |e'|$ and $e \to^* e'$. We can extend Lemma 3 to $\to^*$ by a simple induction on the length of $e \to^* e'$, hence $[\![e']\!]^{\beta pl} \subseteq [\![e]\!]^{\beta pl}$. As $\forall e \in Exp_\bot, |e| \in [\![e]\!]^{\beta pl}$ (by a simple induction on the structure of e), then $t \sqsubseteq |e'| \in [\![e']\!]^{\beta pl} \subseteq [\![e]\!]^{\beta pl}$, hence $t \in [\![e]\!]^{\beta pl}$ by the polarity of Theorem 2. Example 1.1 shows that the converse inclusion does not hold in general.

□

The evident corollary for all these results is the following inclusion chain.

Corollary 4.1. *For any CRWL-program* $\mathcal{P}$*,* $e \in Exp$

$$[\![e]\!]^{sg} \subseteq [\![e]\!]^{rt} \subseteq [\![e]\!]^{\beta pl} \subseteq [\![e]\!]^{\alpha pl}$$

Hence for any $t \in CTerm$*,* $\mathcal{P} \vdash_{CRWL} e \twoheadrightarrow t$ *implies* $\mathcal{P} \vdash e \to^* t$*, which implies* $\mathcal{P} \vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$*, which implies* $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$*.*

Proof. The first part holds just combining Theorems 3, 4, and 5.
Concerning the second part, assume $\vdash_{CRWL} e \twoheadrightarrow t$, in other words, $t \in [\![e]\!]^{sg}$. Then by the first part $t \in [\![e]\!]^{rt}$, hence $e \to^* e'$ such that $t \sqsubseteq |e'|$. But as $t \in CTerm$ it is total and then t is maximal w.r.t. $\sqsubseteq$ (a know property of $\sqsubseteq$ easy to check by induction on the structure of expressions), and so $t \sqsubseteq |e'|$ implies $t \equiv |e'|$, which implies $t \equiv e'$, as t is total (easy to check by induction on the structure of t). Therefore $e \to^* e' \equiv t \in CTerm$, which implies $t \in [\![e]\!]^{rt}$ by definition, as for c-terms t we have $t \sqsubseteq t \equiv |t|$ (a property of shells proved by induction on the structure of t), but then $t \in [\![e]\!]^{\beta pl} \subseteq [\![e]\!]^{\alpha pl}$ by the first part, and so both $\vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$ and $\vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$. □

4.2 Restricted equivalence of $\pi^\alpha CRWL$ and $\pi^\beta CRWL$

In this section we will present a class of programs for which $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ behave the same, thus yielding exactly the same denotation for any expression. In the previous section we saw that $[\![e]\!]^{\beta pl} \subseteq [\![e]\!]^{\alpha pl}$ for any expression and program, therefore we just have to find a class of programs such that $[\![e]\!]^{\alpha pl} \subseteq [\![e]\!]^{\beta pl}$ also holds for programs in that class.

The intuitions and ideas behind the characterization of that class of programs come from Example 3.5 (page 11). The program used there contains two functions f and h defined by the rules $\{f(c(X)) \to d(X,X), h(d(X,Y)) \to d(X,X)\}$, with $d \in CS^2$, under which it is easy to check that $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ behave the same for the expressions $f(c(0)\ ?\ c(1))$ and $h(d(0,0)\ ?\ d(1,1))$.

- Regarding $f(c(0)\ ?\ c(1))$, it is pretty natural for both plural semantics to behave the same, as no wrong information mixup can be performed when combining two substitutions with singleton domain, like $[X/0]$ and $[X/1]$, coming when evaluating $c(0)\ ?\ c(1)$ to get an instance of $c(X)$.
- The case for $h(d(0,0)\ ?\ d(1,1))$ is more surprising at a first look, because then we can obtain the matching substitutions $[X/0, Y/0]$ and $[X/1, Y/1]$, which cannot be safely combined because the set $\{[X/0, Y/0], [X/1, Y/1]\}$ is not compressible. But, as seen in Example 3.5, this poses no problem, because the wrongly intermingled substitution $[X/0\ ?\ 1, Y/0\ ?\ 1]$ used by $\pi^\alpha CRWL$ has the same effect over the right-hand side $d(X,X)$ of the rule for h as the substitution $[X/0\ ?\ 1, Y/\ \bot]$, that can be obtained from combining the compressible set $\{[X/0, Y/\ \bot], [X/1, Y/\ \bot]\}$. This compressible set not only can be used for parameter passing by $\pi^\beta CRWL$, but also can be generated by evaluating the arguments of $h(d(0,0)\ ?\ d(1,1))$ to get an instance of the left-hand side of the rule for h, as $[X/0, Y/\ \bot] \sqsubseteq [X/0, Y/0]$ and $[X/1, Y/\ \bot] \sqsubseteq [X/1, Y/1]$.

What the functions f and h have in common is that, for each argument of the left-hand side of each of their program rules, at most one variable in that argument appears also in the right-hand side. If we only have to care about one variable then we can lower to $\bot$ the value obtained for the other variables in the matching substitution, thus getting a smaller—w.r.t. to $\sqsubseteq$—matching substitution corresponding to a smaller value, that then can be computed thanks to the polarity of $\pi^\alpha CRWL$ from Proposition 1. The effect of this is that we would get a compressible substitution that can be used by $\pi^\alpha CRWL$ to turn a $\mathbf{POR}^\alpha$ step using a possibly non-compressible substitutions into a $\mathbf{POR}^\alpha$ step using a compressible substitutions, that would be then a valid $\mathbf{POR}^\beta$ step as well. Note that in this case extra variables pose no problem, as the only difference between $\mathbf{POR}^\alpha$ and $\mathbf{POR}^\beta$ is the way they handle the matching substitutions obtained by the evaluation of function arguments. Then, as extra variables are instantiated freely and independently of the matching substituions, they always behave the same both under $\pi^\alpha CRWL$ and $\pi^\beta CRWL$.

In the following definition we formally define the class $\mathcal{C}^{\alpha\beta}$ of programs in which the ideas above are materialized.

Definition 5 (Class of programs $\mathcal{C}^{\alpha\beta}$)**.** *The class of programs $\mathcal{C}^{\alpha\beta}$ is defined by*

$$\mathcal{P} \in \mathcal{C}^{\alpha\beta} \text{ iff } \forall (f(p_1, \ldots, p_n) \to r) \in \mathcal{P}. \forall i \in \{1, \ldots, n\}. \#(var(p_i) \cap var(r)) \leq 1$$

where, given a set S, $\#(S)$ stands for the cardinality of S. Note that any program rule in which every argument in its left-hand side is ground or a variable passes the test that characterizes $\mathcal{C}^{\alpha\beta}$: for ground arguments no parameter passing is performed, only matching, so we conjecture that if the arguments in the left-hand side of each program rule are ground then both $CRWL$, term rewriting, $\pi^\alpha CRWL$, and $\pi^\beta CRWL$ behave the same; on the other hand, for variable arguments we have the converse situation so matching is trivial and parameter passing is the important thing, so we conjecture that if the arguments in the left-hand side of each program rule are variables then both term rewriting, $\pi^\alpha CRWL$, and $\pi^\beta CRWL$ behave

the same—*CRWL* remains as the smaller semantics in this case, just consider the program $\{pair(X) \to d(X,X)\}$ and the expression $pair(0\ ?\ 1)$ for which $d(0,1)$ cannot be computed by *CRWL* but it can be by any of the other three semantics.

Anyway, the class $\mathcal{C}^{\alpha\beta}$ is defined by a simple syntactic criterion, which can be easily implemented in any mechanized program analysis tool, and that we have implemented in our prototype from Section 5.

The following theorem formalizes the expected equivalence between $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$ for programs in the class $\mathcal{C}^{\alpha\beta}$.

Theorem 6 (Equivalence of $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$ for the class $\mathcal{C}^{\alpha\beta}$)**.** *For any program $\mathcal{P} \in \mathcal{C}^{\alpha\beta}$, $e \in Exp_{\perp}$*

$$[\![e]\!]^{\alpha pl}_{\mathcal{P}} = [\![e]\!]^{\beta pl}_{\mathcal{P}}$$

This equivalence between $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$ will be very useful for us for several reasons. First of all, as we will see in Section 4.3, $\pi^{\alpha}CRWL$ can be simulated by term rewriting through a simple program transformation, which implies that the same transformation can be used to simulate $\pi^{\beta}CRWL$ for the class of programs $\mathcal{C}^{\alpha\beta}$, thanks to the equivalence from Theorem 6. On the other hand the class $\mathcal{C}^{\alpha\beta}$ is defined by a simple syntactic criterion, which allows its application to mechanized program analysis. Finally, this equivalence grows in importance after realising that the class $\mathcal{C}^{\alpha\beta}$ contains many relevant programs: as a matter of fact all the programs considered in Section 5—where we explore the expressive capabilities of our plural semantics—belong to the class $\mathcal{C}^{\alpha\beta}$.

4.3 Simulating plural semantics with term rewriting

In [33, 35, 30] it was shown that neither *CRWL* can be simulated by term rewriting with a simple program transformation, nor vice versa. Nevertheless, $\pi^{\alpha}CRWL$ can be simulated by term rewriting using the transformation presented in the current section, which can then be used as the basis for a first implementation of $\pi^{\alpha}CRWL$. First we will present a naive version of this transformation, and show its adequacy; later we will propose some simple optimizations for it.

In this section we will restrict ourselves to programs not containing extra variables, i.e., such that for any program rule $l \to r$ we have that $var(r) \subseteq var(l)$ holds, a restriction usually adopted in texts devoted to term rewriting systems [7, 50] for which term rewriting with extra variables is normally considered as an extension of standard term rewriting. Besides, in practical implementations extra variables are usually handled by using narrowing [36, 26] or additional conditions to restrict their possible instantiations [14], in order to avoid a state space explosion in the search process. Therefore we leave the extension of our work to completely deal with extra variables as a subject of future work.

4.3.1 A simple transformation

The main idea in our transformation is to postpone the pattern matching process in order to prevent an early resolution of non-determinism. Instead of presenting the transformation directly, we will first illustrate this concept by applying the transformation over the program $\mathcal{P} = \{f(c(X)) \to d(X,X)\}$ from Example 1.1, which results in the following program $\hat{\mathcal{P}}$.

$$\begin{array}{ll} \hat{\mathcal{P}} = \{ & f(Y) \to \mathit{if}\ \mathit{match}(Y)\ \mathit{then}\ d(\mathit{project}(Y), \mathit{project}(Y)), \\ & \mathit{match}(c(X)) \to \mathit{true}, \mathit{project}(c(X)) \to X \ \ \} \end{array}$$

In the resulting program $\hat{\mathcal{P}}$ the only rule for function f has been transformed so matching is transferred from the left-hand side to the right-hand side of the rule, by means of the auxiliary functions *match* and *project*. As a consequence, when we evaluate by term rewriting under $\hat{\mathcal{P}}$ the function call to f, in the expression $f(c(0)\ ?\ c(1))$ we are not forced anymore to solve the non-deterministic choice between $c(0)$ and $c(1)$ before parameter passing, because any expression matches the variable pattern Y. Therefore the term rewriting step

$$f(c(0)\ ?\ c(1)) \to \mathit{if}\ \mathit{match}(c(0)\ ?\ c(1))\ \mathit{then}\ d(\mathit{project}(c(0)\ ?\ c(1)), \mathit{project}(c(0)\ ?\ c(1)))$$

is sound, thus replicating the argument of f freely without demanding any evaluation, this way keeping its $\pi^{\alpha}CRWL$-denotation untouched: this is the key to achieve completeness w.r.t. $\pi^{\alpha}CRWL$. Note that the guard *if* $match(c(0)\ ?\ c(1))$ is needed to ensure that at least one of the values of the argument matches the original pattern, otherwise the soundness of the step could not be granted. For example if we drop this condition in the translation of the rule '$null(nil) \to true$' for defining an emptiness test for the classical

representation of lists in functional programming, we would get '$null(Y) \to true$', which is clearly unsound because it allows us to rewrite $null(cons(0, nil))$ into $true$. Later on, after resolving the guard, different evaluations of the occurrences of $project(c(0)\ ?\ c(1))$ will solve the non-deterministic choice implied by ?, and project the argument of c, thus leading us to the final values $d(0,0)$, $d(1,1)$, $d(0,1)$, and $d(1,0)$, which are the expected values for the expression in the original program under $\pi^{\alpha}CRWL$.

In the following definition we formalize the transformation by means of the function pST, which for any program rule returns a rule to replace it, and a set of auxiliary *match* and *project* rules for the replacement.

Definition 6 ($\pi^{\alpha}CRWL$ to term rewriting transformation, simple version)**.** *Given a program $\mathcal{P}$, our transformation proceeds rule by rule. For every program rule $(f(p_1, \ldots, p_n) \to r) \in \mathcal{P}$ such that $f \notin \{?, if\ then\}$ we define its transformation as:*

$$pST(f(p_1, \ldots, p_n) \to r)$$
$$= f(Y_1, \ldots, Y_n) \to if\ \ match(Y_1, \ldots, Y_n)\ then\ r[\overline{X_{ij}/project_{ij}(Y_i)}]$$

where
- $\forall i \in \{1, \ldots, n\}$, $\{X_{i1}, \ldots, X_{ik_i}\} = var(p_i) \cap var(r)$ *and* $Y_i \in \mathcal{V}$ *is fresh.*
- $match \in FS^n$ *is a fresh function defined by the rule* $match(p_1, \ldots, p_n) \to true$.
- *Each* $project_{ij} \in FS^1$ *is a fresh symbol defined by the single rule* $project_{ij}(p_i) \to X_{ij}$.

For $f \in \{?, if\ then\}$ the transformation leaves its rules untouched.

It is easy to check that if we use the program $\mathcal{P}$ from Example 1.1 as input for this transformation then it outputs the program $\hat{\mathcal{P}}$ from the discussion above, under which we can perform the following term rewriting derivation.

$$\underline{f(c(0)?c(1))} \to if\ \ \underline{match(c(0)?c(1))}\ then\ d(project(c(0)?c(1)), project(c(0)?c(1)))$$
$$\to^* if\ true\ then\ d(\underline{project(c(0)?c(1))}, project(c(0)?c(1)))$$
$$\to d(project(\underline{c(0)?c(1)}), project(\underline{c(0)?c(1)})) \to^* d(\underline{project(c(0))}, \underline{project(c(1))}) \to^* d(0,1)$$

We do not only claim that this transformation is sound, but also have technical results about the strong adequacy of our transformation $pST(\_)$ for simulating the $\pi^{\alpha}CRWL$ logic using term rewriting. The first one is a soundness result, stating that if we rewrite an expression under the transformed program then we cannot get more results that those we can get in $\pi^{\alpha}CRWL$ under the original program.

Theorem 7. *For any CRWL-program $\mathcal{P}$, and any $e \in Exp_{\bot}$ built up on the signature of $\mathcal{P}$, we have*

$$[\![e]\!]^{\alpha pl}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{\alpha pl}_{\mathcal{P}}$$

As a consequence $[\![e]\!]^{rt}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{\alpha pl}_{\mathcal{P}}$.

Proof (sketch). The first part states the soundness within $\pi^{\alpha}CRWL$ of the transformation. Assuming a $\pi^{\alpha}CRWL$-proof for a statement $pST(\mathcal{P}) \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$ for some $t \in CTerm_{\bot}$, we can then build another $\pi^{\alpha}CRWL$-proof for $\mathcal{P} \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$, by induction on the size of the starting proof—measured as the number of rules of $\pi^{\alpha}CRWL$ used. Full details for that proof can be found in Appendix A (page 48).

Concerning the second part, it follows from combining the first part with Corollary 4.1, because then we can chain $[\![e]\!]^{rt}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{\alpha pl}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{\alpha pl}_{\mathcal{P}}$. □

Regarding completeness of the transformation we have obtained the following result stating that, for any expression one can build in the original program, we can refine by term rewriting under the transformed program any value computed for that expression by $\pi^{\alpha}CRWL$ under the original program.

Theorem 8. *For any CRWL-program $\mathcal{P}$, and any $e \in Exp, t \in CTerm_{\bot}$ built up on the signature of $\mathcal{P}$, if $\mathcal{P} \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$ then exists some $e' \in Exp$ built using symbols of the signature of $pST(\mathcal{P})$ such that $pST(\mathcal{P}) \vdash e \to^* e'$ and $t \sqsubseteq |e'|$. In other words, $[\![e]\!]^{\alpha pl}_{\mathcal{P}} \subseteq [\![e]\!]^{rt}_{pST(\mathcal{P})}$.*

The proof for this result is technically very involved. First of all we have to slightly generalize Theorem 8 to consider not only the functions of the original program but also the auxiliary *match* and *project* functions generated by the transformation, in order to obtain strong enough induction hypothesis.

Lemma 8. *Given a CRWL-program $\mathcal{P}$ let $\hat{\mathcal{P}} \uplus \mathcal{M} = pST(\mathcal{P})$, where $\mathcal{M}$ is the set containing the rules for the new functions match and project, and $\hat{\mathcal{P}}$ contains the new versions of the original rules of $\mathcal{P}$—note that by an abuse of notation the rules for ?, if then presented in Section 2.1 belong implictly both to $\mathcal{P} \uplus \mathcal{M}$ and $\hat{\mathcal{P}} \uplus \mathcal{M}$.*

Then for any $e \in Exp_\bot, t \in CTerm_\bot$ constructed using just symbols in the signature of $\mathcal{P} \uplus \mathcal{M}$ we have $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ implies $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \rightarrow^* e'$ such that $t \sqsubseteq |e'|$.

The proof for Lemma 8 is pretty complicated and it relies on several auxiliary notions, a fully detailed proof can be found in Appendix A. Then Theorem 8 follows as an almost trivial consequence of Lemma 8.

Proof for Theorem 8. Let $\hat{\mathcal{P}} \uplus \mathcal{M} = pST(\mathcal{P})$ be, where $\mathcal{M}$ is the set containing the rules for the new functions *match* and *project*, and $\hat{\mathcal{P}}$ contains the new versions of the original rules of $\mathcal{P}$.

If $e \in Exp, t \in CTerm_\bot$ are built using symbols on the signature of $\mathcal{P}$, then $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ implies $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$, which implies $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \rightarrow^* e'$ such that $t \sqsubseteq |e'|$ by Lemma 8, that is, $pST(\mathcal{P}) \vdash e \rightarrow^* e'$. □

To conclude, the following corollary summarizes the adequacy of the simulation performed by our program transformation.

Corollary 4.2 (Adequacy of $pST(\_)$ for simulating $\pi^\alpha CRWL$)**.** *For any program $\mathcal{P}$, $e \in Exp$ built using symbols of the signature of $\mathcal{P}$*

$$[\![e]\!]^{\alpha pl}_{\mathcal{P}} = [\![e]\!]^{rt}_{pST(\mathcal{P})}$$

Hence $\forall t \in CTerm$ we have that $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ iff $pST(\mathcal{P}) \vdash e \rightarrow^* t$.

Proof. The fist part holds by a combination of Theorem 7 and Theorem 8.
For the second part, if $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ then $t \in [\![e]\!]^{\alpha pl}{}_{\mathcal{P}} = [\![e]\!]^{rt}_{pST(\mathcal{P})}$ by the first part, hence $\exists e' \in Exp$ such that $pST(\mathcal{P}) \vdash e \rightarrow^* e'$ and $t \sqsubseteq |e'|$. But as $t \in CTerm$ then t is maximal w.r.t. $\sqsubseteq$ and so $t \equiv |e'|$ which implies $t \equiv e'$ (these are known properties of shells and $\sqsubseteq$), therefore $pST(\mathcal{P}) \vdash e \rightarrow^* e' \equiv t$. On the other hand if $pST(\mathcal{P}) \vdash e \rightarrow^* t$ then as $t \sqsubseteq t \equiv |t|$ (again because t is a total c-term) we have $t \in [\![e]\!]^{rt}_{pST(\mathcal{P})} = [\![e]\!]^{\alpha pl}_{\mathcal{P}}$, and so $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$. □

As promised at the end of the previous subsection, we can now use the restricted equivalence between $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ from Theorem 6 to extend the adequacy results of the simulation of $\pi^\alpha CRWL$ with term rewriting to $\pi^\beta CRWL$, for the class of programs $\mathcal{C}^{\alpha\beta}$.

Corollary 4.3 (Restricted adequacy of $pST(\_)$ for simulating $\pi^\beta CRWL$)**.** *For any program $\mathcal{P} \in \mathcal{C}^{\alpha\beta}$, $e \in Exp$ built using symbols of the signature of $\mathcal{P}$*

$$[\![e]\!]^{\beta pl}_{\mathcal{P}} = [\![e]\!]^{rt}_{pST(\mathcal{P})}$$

Hence $\forall t \in CTerm$ we have that $\vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$ iff $pST(\mathcal{P}) \vdash e \rightarrow^* t$.

Proof. A straightforward combination of Corollary 4.2 and Theorem 6. □

This last result illustrates the interest of $\pi^\alpha CRWL$. Because of its simplicity, $\pi^\alpha CRWL$ sometimes combines matching substitutions in a wrong way, but it is precisely that same simplicity which allows it to be simulated by term rewriting through a simple program transformation. As a result we can use any available implementation of term rewriting, like the Maude system, to devise an implementation of $\pi^\alpha CRWL$. Besides, thanks to the restricted equivalence between $\pi^\alpha CRWL$ and $\pi^\beta CRWL$, that would also be an implementation of $\pi^\beta CRWL$ for the class $\mathcal{C}^{\alpha\beta}$, and the membership check of program to the class $\mathcal{C}^{\alpha\beta}$ could be also mechanized, because $\mathcal{C}^{\alpha\beta}$ is defined by a simple syntactic criterion. We will see how these ideas are developed in the next sections, where a Maude-based implementation of our plural semantics is presented, and the interest of the class $\mathcal{C}^{\alpha\beta}$ is illustrated.

4.3.2 An optimized transformation

As we already mentioned in our comments after presenting the class $\mathcal{C}^{\alpha\beta}$ in Definition 5 (page 18), we expect that for ground or variable arguments run-time choice and our plural semantics behave the same. We can take advantage of this for applying some optimizations to the program transformation from Definition 6.

- When applied to $null(nil) \to true$, the transformation returns the rules $\{null(Y) \to \textit{if } match(Y) \textit{ then } true,\ match(nil) \to true\}$, which behave the same as the as the original rule. The conclusion is that when a given pattern is ground then no parameter passing will be done for that pattern, and thus no transformation is needed.
- Something similar happens with $pair(X) \to d(X, X)$ for which $\{pair(Y) \to \textit{if } match(Y) \textit{ then } d(project(Y), project(Y)), match(X) \to true, project(X) \to X\}$ is returned. In this case the pattern is a variable to which any expression matches without any evaluation, and the projection functions are trivial, so no transformation is needed neither.

We can apply these ideas to get the following refinement of our original program transformation.

Definition 7 ($\pi^{\alpha}CRWL$ to term rewriting transformation, optimized version)**.** *Given a program $\mathcal{P}$, our transformation proceeds rule by rule. For every program rule $(f(p_1, \ldots, p_n) \to r) \in \mathcal{P}$ we define its transformation as:*

$$pST(f(p_1, \ldots, p_n) \to r)$$
$$= \begin{cases} f(p_1, \ldots, p_n) \to r & \textit{if } \rho_1 \ldots \rho_m \textit{ is empty} \\ f(\tau(p_1), \ldots, \tau(p_n)) \to \textit{if } match(Y_1, \ldots, Y_m) \textit{ then } r[\overline{X_{ij}/project_{ij}(Y_i)}] & \textit{otherwise} \end{cases}$$

where $\rho_1 \ldots \rho_m = p_1 \ldots p_n \mid \lambda p.(p \notin \mathcal{V} \wedge var(p) \neq \emptyset)$.
- $\forall \rho_i$, $\{X_{i1}, \ldots, X_{ik_i}\} = var(\rho_i) \cap var(r)$ and $Y_i \in \mathcal{V}$ is fresh.
- $\tau : CTerm \to CTerm$ is defined by $\tau(p) = p$ if $p \notin \{\rho_1, \ldots, \rho_m\}$; otherwise $\tau(\rho_i) = Y_i$.
- $match \in FS^m$ fresh is defined by the rule $match(\rho_1, \ldots, \rho_m) \to true$.
- Each $project_{ij} \in FS^1$ is a fresh symbol defined by the rule $project_{ij}(\rho_i) \to X_{ij}$.

Note that this transformation is well defined because each $\rho_i \in \{\rho_1, \ldots, \rho_m\}$ contains at least one variable, and so it can be distinguished from any other ρ_j by using syntactic equality thanks to left linearity of program rules, therefore τ is well defined.

We will not give any formal proof for the adequacy of this optimized transformation. Nevertheless note how this transformation leaves untouched the rules for ? and *if then* without defining a special case for them. As the simple transformation from Definition 6 worked well for that rules, that suggests that we are doing the right thing.

We end this section with an example application of the optimized transformation, over the program from Example 3.3. As expected the transformed program behaves under term rewriting like the original one under $\pi^{\alpha}CRWL$.

Example 4.1. *The only rule modified is the one for find, for which we get the following program*

$$\begin{array}{l} \{find(Y) \to \textit{if } match(Y) \textit{ then } (project(Y), project(Y)), \\ \ match(e(N, G, clerk)) \to true, \\ \ project(e(N, G, clerk)) \to N\} \end{array}$$

under which we can perform this term rewriting derivation for twoclerks

$$\begin{array}{ll} & \underline{twoclerks} \to \underline{find(employees(branches))} \\ \to & \textit{if } match(\underline{employees(branches)}) \\ & \quad \textit{then } (project(employees(branches)), project(employees(branches))) \\ \to^* & \underline{\textit{if } match(e(pepe, man, clerk))} \\ & \quad \underline{\textit{then } (project(employees(branches)), project(employees(branches)))} \\ \to^* & (project(\underline{employees(branches)}), project(\underline{employees(branches)})) \\ \to^* & (\underline{project(e(pepe, man, clerk))}, \underline{project(e(maria, woman, clerk))}) \\ \to^* & (pepe, maria) \end{array}$$

5 Programming with singular and plural functions

So far we have presented two novel proposals for the semantics of lazy non-determi-nistic functions, studied some of its properties, and explored their relation to previous proposals like call-time choice and run-time choice. Nevertheless, we have seen just a couple of program examples using the semantics, so

until now we have hardly tested the way we can exploit the new expressive capabilities offered by our plural semantics to improve the declarative flavour of programs. The present section is devoted to the exploration of those expressive capabilities by means of several programs that try to illustrate the virtues of our new plural semantics.

In [43] the authors already explored the capabilities of $\pi^{\alpha}CRWL$ by using the Maude system [14] to develop an interpreter for this semantics based on the program transformation from Section 4.3. The resulting interpreter was then used for experimenting with $\pi^{\alpha}CRWL$, showing how it allows an elegant encoding of some problems, in particular those with an implicit manipulation of sets of values. However, call-time choice still remains the best option for many common programming patterns [23, 3], and this is the reason why it is the semantic option adopted by modern functional-logic programming systems like Toy [36] or Curry [26]. Therefore it would be nice to have a language in which both options could be available. In this section we propose such a language, where the user has the possibility to specify which arguments of each function symbol will be considered "plural arguments." These arguments will be evaluated using our plural semantics, which intuitively means that they will be treated like sets of elements of the corresponding type[8] instead of single elements, while the others will be evaluated under the usual singular/call-time choice semantics traditionally adopted for FLP. Thereby in [44] we extended our Maude-based prototype to support this combination of singular and plural arguments, and used it to develop and test several programs that we think are significant examples of the possibilities of the combined semantics. The source code for these examples and the interpreter to test them can be found at `http://gpd.sip.ucm.es/PluralSemantics`.

As we have two different plural semantics available, then we get two different semantics resulting from their combination with call-time choice, that we have precisely formalized by means of two novel variants of *CRWL* called $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$, corresponding to the combination of call-time choice with $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$, respectively. Our prototype is based on the program transformation from Section 4.3, therefore it is an implementation of $CRWL^{\sigma}_{\pi^{\alpha}}$, and so $CRWL^{\sigma}_{\pi^{\beta}}$ is only supported for programs in the class $\mathcal{C}^{\alpha\beta}$ described in Section 4.2. After those calculus, we introduce the concrete syntax of our interpreter and motivate the combination of singular and plural semantics with a simple example, while the next examples illustrate how to combine singular and plural arguments in depth. Then, after a short discussion about the use of singular and plural arguments, we conclude this section with and a brief outline of the implementation of our prototype.

5.1 The logics $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$

We assume a mapping $\mathit{plurality} : FS \to \{sg, pl\}^*$ called *plurality map* such that, for every $f \in FS^n$, $\mathit{plurality}(f) = b_1 \dots b_n$ fixes its plurality behaviour: if $b_i = sg$ then the i-th argument of f will be interpreted with a singular semantics, otherwise it will be interpreted under a plural semantics. In this line $\mathit{sgArgs}(f) = \{i \in \{1, \dots, ar(f)\} \mid \mathit{plurality}(f)[i] = sg\}$ and $\mathit{plArgs}(f) = \{i \in \{1, \dots, ar(f)\} \mid \mathit{plurality}(f)[i] = pl\}$ are the sets of singular and plural arguments of some $f \in FS$. In particular we say that f is a *singular function* if $\mathit{sgArgs}(f) = \{1, \dots, ar(f)\}$ and that it is a *plural function* when $\mathit{plArgs}(f) = \{1, \dots, ar(f)\}$. A related notion is that of singular and plural variables of a pattern: $\mathit{sgVars}(f(\overline{p})) = \bigcup_{i \in \mathit{sgArgs}(f)} \mathit{var}(p_i)$ and $\mathit{plVars}(f(\overline{p})) = \bigcup_{i \in \mathit{plArgs}(f)} \mathit{var}(p_i)$.

Thus we employ the plurality map to express which function arguments are considered singular arguments and which plural arguments. With this at hand we now define the combined semantics $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$ as the result of taking the rules of *CRWL* and replacing the rule **OR** by either the rule $\mathbf{OR}^{\sigma}_{\pi^{\alpha}}$ or $\mathbf{OR}^{\sigma}_{\pi^{\beta}}$ from Figure 4, respectively. As any variant of *CRWL*, these calculi derive reduction statements of the form $\mathcal{P} \vdash_{CRWL^{\sigma}_{\pi^{\alpha}}} e \twoheadrightarrow t$ and $\mathcal{P} \vdash_{CRWL^{\sigma}_{\pi^{\beta}}} e \twoheadrightarrow t$ that express that t is (or approximates to) a possible value for e in $CRWL^{\sigma}_{\pi^{\alpha}}$ or $CRWL^{\sigma}_{\pi^{\beta}}$, respectively, under the program $\mathcal{P}$. The denotations $[\![e]\!]^{s\alpha p}_{\mathcal{P}}$ and $[\![e]\!]^{s\beta p}_{\mathcal{P}}$ established by these semantics are defined as usual—see Definition 2 (page 15).

Just like in $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$, we consider sets of partial values for parameter passing instead of single partial values, but the novelty is that now these sets are forced to be singleton for singular arguments. This is reflected in the new rules $\mathbf{OR}^{\sigma}_{\pi^{\alpha}}$ and $\mathbf{OR}^{\sigma}_{\pi^{\beta}}$, corresponding to $\mathbf{POR}^{\alpha}$ and $\mathbf{POR}^{\beta}$ respectively, that now have been tuned to take account of the plurality map, as for singular arguments we are only allowed to compute a single value, thus performing parameter passing over it with a substitution from $CSubst_{\perp}$ (as obviously $?\{\theta\} = \theta$), and achieving a singular behaviour (call-time choice).

Example 5.1. *Consider the program* $\{f(X, c(Y)) \to d(X, X,\ Y, Y)\}$ *and a plurality map such that*

[8] As types are not considered through this work here we mean the type naturally intended by the programmer.

$$\mathbf{OR}^{\sigma}_{\pi^{\alpha}}\quad \frac{\begin{array}{ccc} e_1 \twoheadrightarrow p_1\theta_{11} & & e_n \twoheadrightarrow p_n\theta_{n1} \\ \cdots & \cdots & \cdots \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & e_n \twoheadrightarrow p_n\theta_{nm_n} \quad r\theta \twoheadrightarrow t \end{array}}{f(e_1,\ldots,e_n) \twoheadrightarrow t}$$

if $(f(\overline{p}) \to r) \in \mathcal{P}$, $\forall i\ \Theta_i = \{\theta_{i1},\ldots,\theta_{im_i}\}$
$\theta = (\biguplus ?\Theta_i) \uplus \theta_e, \forall i,j\ dom(\theta_{ij}) \subseteq var(p_i)$
$dom(\theta_e) \subseteq vExtra(f(\overline{p}) \to r), \theta_e \in CSubst^{?}_{\perp}$
$\forall i\ m_i > 0, \forall i \in sgArgs(f).m_i = 1$

$$\mathbf{OR}^{\sigma}_{\pi^{\beta}}\quad \frac{\begin{array}{ccc} e_1 \twoheadrightarrow p_1\theta_{11} & & e_n \twoheadrightarrow p_n\theta_{n1} \\ \cdots & \cdots & \cdots \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & e_n \twoheadrightarrow p_n\theta_{nm_n} \quad r\theta \twoheadrightarrow t \end{array}}{f(e_1,\ldots,e_n) \twoheadrightarrow t}$$

if $(f(\overline{p}) \to r) \in \mathcal{P}$, $\forall i\ \Theta_i = \{\theta_{i1},\ldots,\theta_{im_i}\}$ is compressible
$\theta = (\biguplus ?\Theta_i) \uplus \theta_e, \forall i,j\ dom(\theta_{ij}) \subseteq var(p_i)$
$dom(\theta_e) \subseteq vExtra(f(\overline{p}) \to r), \theta_e \in CSubst^{?}_{\perp}$
$\forall i\ m_i > 0, \forall i \in sgArgs(f).m_i = 1$

Figure 4: The rules $\mathbf{OR}^{\sigma}_{\pi^{\alpha}}$ and $\mathbf{OR}^{\sigma}_{\pi^{\beta}}$

plurality(f) = *sg pl*. *The following is a CRWL*$^{\sigma}_{\pi^{\alpha}}$*-proof for the statement* $f(0\ ?\ 1, c(0)\ ?\ c(1)) \twoheadrightarrow d(0,0,0,1)$ *(some steps have been omitted for the sake of conciseness).*

$$\dfrac{\dfrac{\dfrac{(*)}{c(0)\ ?\ c(1) \twoheadrightarrow c(0)}}{\dfrac{\cdots}{c(0)\ ?\ c(1) \twoheadrightarrow c(1)}} \quad \dfrac{\dfrac{\cdots}{0 \twoheadrightarrow 0} \quad \dfrac{\cdots}{0 \twoheadrightarrow 0} \quad \dfrac{\cdots}{0\ ?\ 1 \twoheadrightarrow 0} \quad \dfrac{\cdots}{0\ ?\ 1 \twoheadrightarrow 1}}{d(0,0,0\ ?\ 1,0\ ?\ 1) \twoheadrightarrow d(0,0,0,1)}\ \mathbf{DC}}{f(0\ ?\ 1, c(0)\ ?\ c(1)) \twoheadrightarrow d(0,0,0,1)}\ \boldsymbol{OR}^{\sigma}_{\pi^{\alpha}}$$

where $(*)$ *is the following proof:*

$$\dfrac{\dfrac{\dfrac{}{0 \twoheadrightarrow 0}\ \mathbf{DC}}{c(0) \twoheadrightarrow c(0)}\ \mathbf{DC} \quad \dfrac{}{c(1) \twoheadrightarrow \perp}\ \mathbf{B} \quad \dfrac{\cdots}{c(0) \twoheadrightarrow c(0)}}{c(0)\ ?\ c(1) \twoheadrightarrow c(0)}\ \boldsymbol{OR}^{\sigma}_{\pi^{\alpha}}$$

Note that $d(0,1,0,1)$ *is not a correct value for the expression* $f(0\ ?\ 1, c(0)\ ?\ c(1))$ *under CRWL*$^{\sigma}_{\pi^{\alpha}}$*, because the first argument of* f *is singular and therefore the two occurrences of* X *in the right-hand side of its rule share the same single value, fixed on parameter passing. Besides, as this program is in the class* $\mathcal{C}^{\alpha\beta}$*, then it behaves the same under* π^{α}*CRWL and* π^{β}*CRWL, and therefore also under CRWL*$^{\sigma}_{\pi^{\alpha}}$ *and CRWL*$^{\sigma}_{\pi^{\beta}}$*, so the previous proof and comments also hold for CRWL*$^{\sigma}_{\pi^{\beta}}$*.*

On the other hand if we take the same program and evaluate $f(0\ ?\ 1, c(0)\ ?\ c(1))$ *under term rewriting—which ignores the plurality map—, its behaviour is significantly different:*

$$\begin{array}{l} f(0\ ?\ 1, \underline{c(0)\ ?\ c(1)}) \to \underline{f(0\ ?\ 1, c(0))} \to d(\underline{0\ ?\ 1}, 0\ ?\ 1, 0, 0) \\ \to d(0, \underline{0\ ?\ 1}, 0, 0) \to d(0,1,0,0) \end{array}$$

A first step resolving the choice between $c(0)$ *and* $c(1)$ *is unavoidable in order to get an expression matching for the only rule for* f*, thus for any reachable c-term the last two arguments of* d *will be the same, contrary to what happens in CRWL*$^{\sigma}_{\pi^{\alpha}}$ *and CRWL*$^{\sigma}_{\pi^{\beta}}$ *under the given plurality map. Nevertheless its first two arguments can be different, contrary to what happens under CRWL*$^{\sigma}_{\pi^{\alpha}}$ *and CRWL*$^{\sigma}_{\pi^{\beta}}$*. In conclusion, it is easy to define a program and a plurality map for them such that neither CRWL*$^{\sigma}_{\pi^{\alpha}}$ *nor CRWL*$^{\sigma}_{\pi^{\beta}}$ *are comparable to term rewriting w.r.t. set inclusion of the computed values.*

A useful intuition about programs comes from considering the singular arguments as fixed individual values, while thinking about the plural ones as sets. We could have chosen to specify the plurality or singularity of functions instead of that of its arguments, but the use of arguments with different plurality arises naturally in programs, in the same way it is natural to have arguments of different types. We will illustrate this fact later on by means of several examples.

```
(plural SAMPLE-PROGRAM is
   f is plural .
   f(c(X)) -> p(X, X) .
 endp)
```

Figure 5: Concrete syntax of programs

Regarding properties of these semantics (see Appendix A for more details), both $CRWL^{\sigma}_{\pi^\alpha}$ and $CRWL^{\sigma}_{\pi^\beta}$ inherit the properties of $\pi^\alpha CRWL$ from Section 3.1, for the same reason $\pi^\beta CRWL$ inherits the properties of $\pi^\alpha CRWL$. The most important among these properties is their compositionality, which expresses the value-based philosophy underlying $CRWL^{\sigma}_{\pi^\alpha}$ and $CRWL^{\sigma}_{\pi^\beta}$ "all I know about an expression is its set of values," and that holds for the corresponding reformulation of Theorem 1—as it can be proved by a straightforward modification of the proof for that theorem. Bubbling is also incorrect for both $CRWL^{\sigma}_{\pi^\alpha}$ and $CRWL^{\sigma}_{\pi^\beta}$, just like it happens for $\pi^\alpha CRWL$ and $\pi^\beta CRWL$: in fact Example 3.2 can be reused to prove it. Nevertheless, just like for $\pi^\alpha CRWL$ and $\pi^\beta CRWL$, bubbling is correct for a particular kind of contexts, in this case not only for c-contexts but for the bigger class of *singular contexts* $s\mathcal{C}$, which are contexts whose holes appear only under a nested application of constructor symbols or singular function arguments: $s\mathcal{C} ::= [\,] \mid c(e_1, \ldots, s\mathcal{C}, \ldots, e_n) \mid f(e_1, \ldots, s\mathcal{C}, \ldots, e_n)$, with $c \in CS^n$, $f \in FS^n$ such that the subcontext appears in a singular argument of f, and $e_1, \ldots, e_n \in Exp_\bot$. For singular contexts we get a compositionality result for singular contexts analogous to that of Proposition 3—following the same scheme as the proof for Theorem 1—that can be used to easily prove the correctness of bubbling for singular contexts.

We conclude our discussion about $CRWL^{\sigma}_{\pi^\alpha}$ and $CRWL^{\sigma}_{\pi^\beta}$ with the following result stating that they are in fact conservative extensions of both *CRWL* (call-time choice, or equivalently, singular non-determinism) and their corresponding plural semantics, as it was apparent from their rules.

Theorem 9 (Conservative extension)**.** *Under any program and for any $e \in Exp_\bot$:*

1. *If the program contains no extra variables and every function is singular then $[\![e]\!]^{s\alpha p} = [\![e]\!]^{sg} = [\![e]\!]^{s\beta p}$.*
2. *If every function is plural then $[\![e]\!]^{s\alpha p} = [\![e]\!]^{\alpha pl}$ and $[\![e]\!]^{s\beta p} = [\![e]\!]^{\beta pl}$.*

Proof. If every function is singular and the program containts no extra variables then $\mathbf{OR}^{\sigma}_{\pi^\alpha}$ and $\mathbf{OR}^{\sigma}_{\pi^\beta}$ are equivalent to $\mathbf{OR}$, so both *CRWL*, $CRWL^{\sigma}_{\pi^\alpha}$, and $CRWL^{\sigma}_{\pi^\beta}$ behave the same. Note that the absence of extra variables is essential, as for example from the program $\{f \to d(X, X)\}$ we get $[\![f]\!]^{sg} \not\ni d(0, 1) \in [\![f]\!]^{s\alpha p} = [\![f]\!]^{s\beta p}$.

Similarly, if every function is plural then $\mathbf{OR}^{\sigma}_{\pi^\alpha}$ and $\mathbf{OR}^{\sigma}_{\pi^\beta}$ are equivalent to $\mathbf{POR}^\alpha$ and $\mathbf{POR}^\beta$, respectively. Note that extra variables pose no problem in this case, as any of these plural semantics is able to instantiate them with an arbitrary substitution from $CSubst^?_\bot$. □

5.2 Commands

In this section we introduce the concrete syntax of our language and the commands provided by our interpreter. The system is started by loading in Maude the file `plural.maude`, available at `http://gpd.sip.ucm.es/PluralSemantics`. It starts an input/output loop that allows the user to introduce commands by enclosing them in parens. Programs start with the keyword `plural`, followed by the module name and the keyword `is`, and finish with `endp`, as exemplified in Figure 5. The body of the programs is a list of statements of the form e_1 `->` e_2 `.`, indicating that the program rule $e_1 \to e_2$ is part of the program.

The plurality map is specified by means of `is` annotations for each function of the program. These annotations have the form f `is` *plurality* `.`, where *plurality* can take the values `singular` for singular functions, `plural` for plural functions, or a sequence composed by the characters `s` and `p` specifying in more detail the plurality behaviour for each function argument, along the lines of the beginning of Section 5.1: if the i-th element of this chain is the character `s` then the i-th argument of f will be a singular argument, otherwise it will be considered a plural argument. *Functions are considered singular by default when no* `is` *annotation is provided.*

The system is able to evaluate any expression built with the symbols of the program, under the semantics specified by the $CRWL^{\sigma}_{\pi^\alpha}$ logic. *The prototype does not support programs with extra variables*, for two main reasons. First of all, it is based on the transformation from Section 4.3, whose adequacy

has been only proved for programs without extra variables. But the main reason is the lack of a suitable narrowing mechanism for plural variables, which is the resort usually employed by FLP systems to deal with the space explosion caused by extra variables [25, 36, 24]. We consider the development of a plural narrowing mechanism an interesting subject of future work, but for now and for the rest of the paper, we restrict ourselves to programs not containing extra variables.

The system provides by default the constant c-terms `tt` (for *true*) and `ff` (for *false*), and two more handy functions: the binary function `_?_`, that is used with infix notation, and the `if_then_` function, used with mixfix notation, defined by the following rules:

```
X ? Y -> X .
X ? Y -> Y .
if tt then E -> E .
```

Note that, since no `is` annotation is provided, both functions are singular.

Once a module has been introduced, the user can evaluate expressions with the command:

```
(eval [[depth = DEPTH]] EXPRESSION .)
```

where `EXPRESSION` is the expression to be evaluated and `DEPTH` is a bound in the number of steps. If this last value is omitted, the search is assumed to be unbounded. If the term can be reduced to a c-term, it will be printed and the user can use

```
(more .)
```

until no more solutions are found.

It is also possible to switch between two evaluation strategies, depth-first and breadth-first, with the commands:

```
(depth-first .)
(breadth-first .)
```

Finally, the system can be rebooted with the command

```
(reboot .)
```

5.3 Examples

In this section we show how to use the commands above, by means of two examples.

5.3.1 Clerks

First we show how to implement in our tool the program from Example 3.3 (page 10), slightly extended by adding a new branch to the bank. The different branches are defined by using the non-deterministic function `?`, that here has to be understood as the set union operator. In the same line, for each branch the function `employees` returns the set of its employees:

```
branches -> madrid ? vigo ? badajoz .

employees(madrid) -> e(pepe, men, clerk) ? e(paco, men, boss) .
employees(vigo) -> e(maria, women, clerk) ? e(jaime, men, boss) .
employees(badajoz) -> e(laura, women, clerk) ? e(david, men, clerk) .
```

Now, we define a function `twoclerks` which searches in the database for the names of two employees working as clerks. It calls the function `find`, which has been marked with the keyword `plural` in order to express that its argument will be understood as a set of records from the database of the bank. Therefore, although the same variable `N` is used in the two components of the pair in the right-hand side of its rule, each one can be instantiated with different values:

```
twoclerks -> find(employees(branches)) .
find is plural .
find(e(N,G,clerk)) -> p(N,N) .
```

Once the module has been loaded in our system,[9] we can use the `eval` command to evaluate expressions, and the command `more` to find the next solutions:

```
Maude> load clerks.plural
Module introduced.
Both alpha and beta plural semantics supported for this program.

Maude> (eval twoclerks .)
Result: p(pepe,pepe)

Maude> (more .)
Result: p(pepe,maria)
```

This program works as we expected, even if all the functions are marked as plural (i.e., if $\pi^{\alpha}CRWL$ is used). However it can be improved in several directions. First of all, we are interested in getting two *different* clerks. To do that we will define a function `vals` that generates a list containing different values of its argument. This function will use an auxiliary function `newIns` that appends an element at the beginning of a list ensuring that the remaining elements of the list are different to the new one. This is checked by `diffL`, which returns the list in its second argument when it does not contain its first argument, and otherwise fails. Thus a disequality test is needed, but in our minimal framework we do not dispose of disequality constraints, common in FLP languages [25, 4]. Nevertheless we can implement a ground version of disequality through regular program rules, as it is done here in the function `neq`.

```
newIns is singular .
newIns(X, Xs) -> cons(X, diffL(X, Xs)) .

diffL(X, nil) -> nil .
diffL(X, cons(Y, Xs)) ->
     if neq(X, Y) then cons(Y, diffL(X, Xs)) .

neq(pepe, paco) -> tt .
neq(pepe, maria) -> tt .
...
```

Note that we need `newIns`, `diffL`, and `neq` to be singular because they essentially perform tests, and when performing a test we naturally want the returning value to be the same which has been tested. For example, the following program:

```
isWoman(maria) -> tt .
isWoman(laura) -> tt .
...
filterWomen(P) -> if isWoman(P) then P
```

would have a funny behaviour if `filterWomen` had been declared a plural function, because then for `filterWomen(maria ? pepe)` we could compute `pepe` as a correct value.

On the other hand the function `vals` is marked as *plural* because it is devised to generate lists of different values of its argument. Note the combination of plurality, to obtain more than one value from the argument of `vals`, and singularity, which is needed for the tests performed by `newIns`:

```
vals is plural .
vals(X) -> newIns(X, vals(X)) .
```

We generalize now our search function to look for any number of clerks, not just two. To do that we will use the function `nVals` below, that returns a list of different values corresponding to different evaluations of its second argument. Therefore that second argument has to be declared as plural, while its first argument is singular as it fixes the number of values claimed (that is, the length of the returning list in the Peano notation for natural numbers):

[9] The tool also indicates whether the program belongs to the class $\mathcal{C}^{\alpha\beta}$—remember that in that case $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$ would be equivalent and so both would be supported by the system—or not.

```
nVals is sp .
nVals(N, E) -> take(N, vals(E)) .

take(s(N), cons(X, Xs)) -> cons(X, take(N, Xs)) .
take(z, Xs) -> nil .
```

This `nVals` function is an example of how the use of plural arguments allows us to simulate some features that in a pure call-time choice context have to be defined at the meta level, in this case the `collect` [36] or `findall` [24] primitives of standard FLP systems.

Finally the function `nClerks` starts the search for a number of different clerks specified by the user. It uses the auxiliary function `findClerks`, that returns the name of the clerks:

```
nClerks is singular .
nClerks(N) -> nVals(N, findClerk(employees(branches))) .

findClerk is singular .
findClerk(e(N,G,clerk)) -> N .
```

Now we can search for three different clerks, obtaining `pepe`, `maria`, and `laura` as the first possible result:

```
Maude> (eval nClerks(s(s(s(z)))) .)
Result: cons(pepe,cons(maria,cons(laura,nil)))
```

As anticipated in Example 3.4 (page 10), we can use this technique to solve the problem of finding the names of the clerks paired with their genre, but avoiding the wrong information mixup caused by a purely plural approach using the style of the plural `find` function above, under $\pi^{\alpha}CRWL$. To do that we just have to define a new auxiliary function `findClerksNG` that this time returns a pair composed by the name of the clerk and his or her genre.

```
nClerksNG is singular .
nClerksNG(N) -> nVals(N, findClerkNG(employees(branches))) .

findClerkNG is singular .
findClerkNG(e(N,G,clerk)) -> p(N, G) .
```

The fact that `findClerksNG` is singular, just like `findClerks`, ensures that the names and genres will be correctly paired. Besides, note that the whole Clerks program presented here belongs to the class $\mathcal{C}^{\alpha\beta}$, therefore its evaluation under $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$ is the same and any wrong information mixup is prevented. We can check this by searching again for three different clerks:

```
Maude> (eval nClerksNG(s(s(s(z)))) .)
Result: cons(p(pepe,men),cons(p(maria,women),cons(p(laura,women),nil)))
```

In the next example we will see more clearly how to decide the plurality of functions. Remember that the key idea that singular arguments are used *to fix their values* while plural arguments are needed when we want to use *sets of values*.

5.3.2 Dungeon

Ulysses has been captured and he wants to cheat his guardians using the gold he carries from Troy. Thus, he needs to know whether there is an escape (what we define as obtaining the `key` of its jail) and, if possible, which is the path to freedom (we define each step of this path as a pair composed of a guardian and the item Ulysses obtains from him).

He uses the function `ask` to interchange items and information with his guardians. Since each guardian provides different information we have to assure that they are not mixed, and thus its first argument will be singular; on the other hand he may offer different items to the same guardian, thus the second argument will be plural: this function needs plurality `sp`:

```
ask is sp .
```

The guardians have a complex behaviour, `circe` exchanges Ulysses' `trojan-gold` by either the `sirens-secret` or an `item(treasure-map)`; `calypso`, once she receives the `sirens-secret`, offers the `item(chest-code)`; `aeolus` can `combine` two items;[10] and `polyphemus` gives Ulysses the `key` once he can give him the combination of the `treasure-map` and the `chest-code`:

```
ask(circe, trojan-gold) -> item(treasure-map) ? sirens-secret .
ask(calypso, sirens-secret) -> item(chest-code) .
ask(aeolus, item(M)) -> combine(M,M) .
ask(polyphemus, combine(treasure-map, chest-code)) -> key .
```

In the same line, `askWho` has as arguments a (*fixed*) guardian and a message (probably with many items) for him, so it also has plurality `sp`. This function returns the next step in the Ulysses' path to freedom, that is, a pair with the guardian and the items obtained from him with the function `ask`:

```
askWho is sp .
askWho(Guardian, Message) -> p(Guardian, ask(Guardian, Message)) .
```

The following functions, which are in charge of computing the actions that must be performed in order to escape, are marked as plural because they treat their corresponding arguments as sets of pairs where the second component is an item or some piece of information, and the first one is the actor which provided it. The function `discoverHow` returns the set of pairs of that shape that can be obtained starting from those contained in its argument, and then chatting to the guardians. Hence it returns, either its argument, or the result of exchanging the current information with some guardian and then iterating the process. That exchange is performed with `discStepHow`, that non-deterministically offers some of the items or information available, to one of the guardians:

```
discoverHow is plural .
discoverHow(T) -> T ? discoverHow(discStepHow(T) ? T) .

discStepHow is plural .
discStepHow(p(W, M)) -> askWho(guardians, M) .

guardians -> circe ? calypso ? aeolus ? polyphemus .
```

Note that the additional disjunction `? T` in the recursive call to `discStepHow` is needed in order to be able to combine the old information with the new one resulting after one exchanging step. This point can be illustrated better with the following program:

```
genPairs is plural .
genPairs(P) -> P ? genPairs(genPairsStep(P) ? P) .

genPairsStep is plural .
genPairsStep(P) -> p(P, P) .

genPairsBad is plural .
genPairsBad(P) -> P ? genPairsBad(genPairsStep(P)) .
```

There the functions `genPairs` and `genPairsBad` follow the same pattern as `discoverHow`, but this time are designed to generate values made up with pairs and the supplied argument. Besides these functions share the same "step function" `genPairsStep`. Nevertheless their behaviour is very different, as we can see evaluating the expressions `genPairs(z)` and `genPairsBad(z)`: the point is that the value `p(p(z,z),z)` can be computed for the former but not for the latter, because `z` and `p(z,z)` are values generated in different recursive calls to `genPairsBad`. But this poses no problem for `genPairs`, because the extra `? P` in its definition makes it possible to combine those values.

Finally, the search is started with the function `escapeHow`, that initializes the search with the trojan gold provided by Ulysses:

```
escapeHow -> discoverHow(p(ulysses, trojan-gold)) .
```

Once the module is introduced, we can start the search with the command:

[10] Note that we say *two* items when the function only shows *one*. This rule uses the expressive power of plural semantics to allow the combination of different items.

```
Maude> (eval escapeHow .)
Result: p(ulysses,trojan-gold)
```

When this first result has been computed, we can ask the tool for more with the command `more`, that progressively will show the path followed by Ulysses to escape:

```
Maude> (more .)
Result: p(circe,item(treasure-map))

Maude> (more .)
Result: p(circe,sirens-secret)

Maude> (more .)
Result: p(calypso,item(chest-code))

...

Maude> (more .)
Result: p(polyphemus,key)
```

In this example the function `discoverHow` is an instance of an interesting pattern of plural function: a function that performs deduction by repeatedly combining the information we have fed it with the information it infers in one step of deduction. Therefore in its definition the function `?` has to be understood again as the set union operator, as it is used to add elements to the set of deduced information. On the other hand the use of a singular argument in `askWho` is unavoidable to be able to keep track of the guardian who answers the question, while its second argument has to be plural because it represents the knowledge accumulated so far.

Several variants of this problem can be conceived, in particular currently it is simplified because the items are not lost after each exchange —this is why Ulysses' bag is bottomless. Anyway we think that this version of the problem is relevant because in fact it corresponds to a small model of an intruder analysis for a security protocol, where Ulysses is the intruder, the guardians are the honest principals, the key is the secret and complex behaviours of the principals can be described through the patterns in left-hand sides of program rules. In this case we assume that the intruder is able to store any amount of information, and that this information can be used many times. Nevertheless we also think that different variants of the problem should be tackled in the future, and that the addition of equality and disequality constraints to our framework could be decisive to deal with those problems.

With this program we conclude our presentation of some examples that show the expressive capabilities of our plural semantics. In these examples we have tried to find a way of using $CRWL^{\sigma}_{\pi^{\alpha}}$ for programming, so it could be more than just a semantic eccentricity. Although we have found some interesting uses of our plural semantics, in particular the meta-like function `nVals`, and the deduction programming pattern correspondint to `discoverHow`, we cannot still say that we have found a "killer application" for our plural semantics. Only time will tell if this semantics are useful, because these proposals are still too young to have a reasonable benchmark collection. Our prototype opens the door to experimenting with this new semantics, and in that sense it contributes to the development of such collection. Anyway, we admit that our plural semantics probably will only be useful in some fragments of the programs, and that is why we have proposed to combine it with the usual singular semantics of FLP. As a final remark the reader can check, by hand or by using our prototype, that all the program examples in this section belong to the $\mathcal{C}^{\alpha\beta}$ class—and hence they behave the same both under $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$—, which motivates the relevance of that class of programs. But again, as the collection of examples is very small, this does not gives a strong argument about the usefulness of this class of programs, but just an encouraging indicator.

5.4 Discussion: to be singular or to be plural?

After these examples, we (hopefully) should have some intuitions about how to decide the plurality of function arguments. Our first resort is considering that plural arguments are used to represent sets of values, while singular arguments denote single values. But this does not work for any situation, for example consider the function `findClerk` whose plurality is singular, although its argument intuitively denotes a set of records from the database. On the other hand we may consider that its argument denotes a single record, and that `findClerk` defines how to extract the name from a single employee,

which motivates the final plurality choice. In this case the program behaves the same declaring `findClerk` either singular or plural, because the variables in its arguments are used only once. As a rule of thumb we should try to have as little plural arguments as possible, because these arguments increase the search space more than the singular ones, as using a plural semantics we can compute more values than under a singular semantics, as seen in Section 4.1. Hence in this case it is better to declare `findClerk` as singular.

Thus having a more formal criterion about the equivalence of plurality maps would be useful to minimize the search space of our programs and understand them better. A static adaptation of the determinism analysis of [13] could be useful, as it would help us to detect deterministic functions of our programs, for which the plurality map would not matter, as we expect to easily extend the equivalence results of singular/call-time choice and run-time choice for deterministic programs of [33, 30] to our plural semantics. We also should try to develop equational laws about non-determinism. In fact a first step in this line is the discussion about the correctness of bubbling for singular contexts from Section 5.1. Anyway, all these are subjects of future work.

5.5 Implementation

The system described in the previous sections has been implemented in the Maude system [14], a high-level language and high-performance system supporting both equational and rewriting logic computation for a wide range of applications. The fundamental ingredients for this implementation are a core language into which all programs are transformed, and an interpreter for the operational semantics of the core language that is used to execute programs.

The transformation into core language treats each program rule separately and applies two different transformation stages to them. The first one applies a modification of the transformation described in Definition 7, but now taking into account only those arguments marked as *pl* in the plurality map described in Section 5.1. Then in the second stage, which consists in a modification of the sharing transformation of [35, Def. 1], we introduce a let-binding for each singular variable that also appears in the right-hand side, therefore obtaining subexpression sharing, and as a consequence, a singular behaviour for those arguments.

Once source programs have been transformed into core programs, we can execute them by using a heap-based operational semantics for the core language [44]. A heap is just a mapping from variables to expressions that represents a graph structure, as the image of each variable is interpreted as a subgraph definition. The nodes of that implied graph are defined according to those let-bindings introduced by the transformation into core language. The operational semantics manipulates this heap, and contains rules for removing useless bindings, propagating the terms associated to a variable, and for creating new bindings for each of singular argument when their corresponding let-bindings are found. Finally, in order to turn the operational semantics of the core language into an effective operational mechanism for $CRWL^{\sigma}_{\pi\alpha}$, we have adapted the natural rewriting strategy in [18] to deal with these heaps, ensuring that the evaluation is performed on-demand. Both the program transformation into core language, the interpreter, and our adaptation of natural rewriting, have been implemented in Maude with an intensive exploiting of its reflection capabilities, thus obtaining an executable interpreter for $CRWL^{\sigma}_{\pi\alpha}$. More details about our implementation can be found in [44].

We decided to follow the line of employing the transformation from Definition 7 and then using a language that implements non-deterministic term rewriting to run the transformed program, because our motivation was to obtain a simple proof of concept prototype that could be used to experiment with the new semantics, but obtaining an optimized implementation is out of the scope of this work. The addition of the natural rewriting on-demand strategy was neccesary in order to reduce the search space to a reasonable size, but we are aware of other approaches that maybe could improve the efficiency of the system. In particular we could have adapted the techniques in [5, 12, 11] that rely on turning the function $? \in FS^2$ into a constructor in order to explicitly represent non-deterministic computations in a deterministic language, which results in additional advantages like a kind of backtracking memoization called "sharing across non-determinism." That would allow us to use Maude functional modules, which are much more efficient than the non-deterministic system modules that are used in our current implementation. But as functional modules perform eager evaluation, then we should also employ the context-sensitive rewriting [38] features of Maude—offered as `strat` annotations—to get the lazy evaluation that correspond to our semantics. That would entail adapting the techniques from [5, 12, 11] from a call-time choice setting to the run-time choice semantics of term rewriting, and also using the techniques in [37] to introduce the `strat` annotations needed to ensure lazy evaluation. This is a possible roadmap that could be followed in case a more optimized implementation of $CRWL^{\sigma}_{\pi\alpha}$ should be developed. The

monad transformer of [19] is another alternative in the same line, as it also provides a representation of non-determinism with support for memoization in a deterministic language, in this case Haskell, that could be used as the basis for an implementation of $CRWL^{\sigma}_{\pi^{\alpha}}$ by modifying the transformation from FLP programs with call-time choice into Haskell from [10]. As that work is placed in a higher order setting, our plural semantics should be first extended with higher order capabilities, following the line of [21].

6 Concluding remarks and future work

The starting point of this work is the observation that the traditional identification between run-time choice and a plural denotational semantics is wrong in a non-deterministic functional language with pattern matching. To illustrate that, we have provided formulations for two different plural semantics that are different from run-time choice: the $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$ semantics. We argue that the run-time choice semantics induced by term rewriting is not the best option for a value-based programming language like current implementations of FLP because of its lack of compositionality. Nevertheless, our plural semantics are compositional for a simple notion of value—the notion of partial c-term—, just like the usual call-time choice semantics adopted by modern FLP languages, following the value-based philosophy of the FLP paradigm: "all I care about an expression is the set of its values." This, together with the fact that our concrete formulations for these plural semantics are variants of the $CRWL$ logic—a standard formulation for singular/call-time choice semantics in FLP—, turns the problem of devising a combined semantics for singular and plural non-determinism into a trivial task, getting the $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$ logics as a result. The combination of singular and plural semantics in the same language is interesting and follows naturally when programming, as it allows us to reuse known programming patterns from the more usual singular/call-time choice semantics, standard in modern FLP systems, while we are still able to use the new capabilities of the novel plural semantics for some interesting fragments of the program. In these logics, apart from the program, the user may specify for each function which of its arguments will be marked as singular and which as plural, resulting in different parameter passing mechanism. A simple intuition that works in most situations can be considering plural arguments as sets of values and singular arguments as individual values. We have not only proposed such semantic combinations but we have also provided a prototype implementation for $CRWL^{\sigma}_{\pi^{\alpha}}$ using the Maude system (see [43, 44] for details about the implementation), in which the program transformation to simulate $\pi^{\alpha}CRWL$ with term rewriting—a standard formulation for run-time choice—also presented in this work is a crucial ingredient. The resulting system, available at `http://gpd.sip.ucm.es/PluralSemantics`, is an interpreter for the $CRWL^{\sigma}_{\pi^{\alpha}}$ logic that we have used to develop several programs examples that exploit the new expressive capabilities of the combined semantics, in order to improve the declarative flavour of programs.

Along the way we have also made several contributions at the foundational level. We have studied the technical properties of $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$, providing formal proofs for its compositionality and also for other interesting properties like polarity, several monotonicity properties for substitutions, and a restricted form of bubbling for constructor contexts. Then we have compared the different semantics for non-determinism considered in this work w.r.t. the set of computed values, concluding that they form the inclusion chain $CRWL \subseteq$ term rewriting $\subseteq \pi^{\beta}CRWL \subseteq \pi^{\alpha}CRWL$, corresponding to the chain singular/call-time choice $\subseteq$ run-time choice $\subseteq \beta$-plural $\subseteq \alpha$-plural. Besides, we have determined that for the class of programs $\mathcal{C}^{\alpha\beta}$, characterized by a simple syntactic criterion, our plural semantics proposals $\pi^{\alpha}CRWL$ and $\pi^{\beta}CRWL$ are equivalent. We have also provided a formal proof of the adequacy of the (non-optimized version of the) transformation used by our prototype to simulate $\pi^{\alpha}CRWL$ with term rewriting. As a consequence, this transformation can be used to simulate $\pi^{\beta}CRWL$ for programs in the class $\mathcal{C}^{\alpha\beta}$. Regarding the combined semantics $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$, it is easy to see that they inherit the good properties of both $CRWL$, $\pi^{\alpha}CRWL$, and $\pi^{\beta}CRWL$, and we have also proved that the combined semantics are conservative extensions of both singular/call-time choice and their corresponding plural semantics.

These questions were first approached in previous works by the authors [45, 43, 44], however in this paper we do not only give a revised and unified presentation, but we have also included several important novel results.

- All the technical results from those works have been extended to deal with programs with extra variables, except those results regarding the simulation of $\pi^{\alpha}CRWL$ with term rewriting from Section 4.3. The new technical results have been also proved for programs with extra variables. Besides, we have fixed some errata from the original works, in particular the formulation of bubbling for

$\pi^\alpha CRWL$, the definition of the operator ? over sequences of $CSubst_\bot$, and also some other minor mistakes in the proofs. The formulations of bubbling for constructor and singular contexts are novel contributions of this paper.

- The plural semantics $\pi^\beta CRWL$, inspired in the proposal from [9], is introduced in this work for the first time. We give clear explanations of some problematic situations where $\pi^\alpha CRWL$ performs a wrong information mixup, and how our attempts to fix those problems, inspired in the solutions from [9], led us to the current formulation of $\pi^\beta CRWL$, which leans on the notion of compressible set of partial c-substitutions.

- As the formulations of $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ are very similar, it was not difficult to check that $\pi^\beta CRWL$ also enjoys the same basic properties of $\pi^\alpha CRWL$. Nevertheless, it was more difficult to place $\pi^\beta CRWL$ in the semantic inclusion chain from [45], being a key idea the notion of compressible completion of a set of $CSubst_\bot$, and its related results. The characterization of the class of programs $\mathcal{C}^{\alpha\beta}$ for which $\pi^\alpha CRWL$ and $\pi^\beta CRWL$ are equivalent, and the formal proof for that equivalence are also novel, obviously.

- Finally, the logic $CRWL^\sigma_{\pi^\beta}$ is also a novel contribution of this work, but in this case its definition was straightforward, because it follows the same pattern as the definition for $CRWL^\sigma_{\pi^\alpha}$.

Previously to ours, not much work has been done in the combination of singular and plural non-determinism in functional or functional-logic programming, since mainstream approaches [51, 36, 24] only support the usual singular semantics. More close are the combinations of call-time and run-time choice of [35, 32], which anyway follow a different approach as the plural sides of $CRWL^\sigma_{\pi^\alpha}$ and $CRWL^\sigma_{\pi^\beta}$ are essentially different to run-time choice. Anyway, we still think that the combination of call-time choice and run-time choice is not very suitable for value-based languages because of the lack of compositionality for values of run-time choice. The monad transformer of [19], devised to improve the laziness of non-deterministic monads while retaining a call-time choice semantics, is based on a `share` combinator which plays a role similar to the let-bindings of our core language. The authors seem to be interested in staying in a pure call-time choice framework, but maybe a combination of call-time and run-time choice could be achieved there too, getting something similar to [35] but again essentially different to $CRWL^\sigma_{\pi^\alpha}$ and $CRWL^\sigma_{\pi^\beta}$ for the same reason. Besides that work is focused in implementation issues of FLP in concrete deterministic functional languages, while in ours we start from the more abstract world of CS's and are fundamentally concerned in exploring the language design space.

We contemplate several interesting subjects of future work. As pointed in Sections 4.3 and 5.2, the development of a suitable plural narrowing mechanism would be the key for finding an effective way of handling extra and free variables. Besides, in our examples it has arisen the necessity of equality and disequality constraints (whose ground versions have been simulated by using regular functions), that will ease and shorten the definition of programs, and increase the expressiveness of the setting. Both subjects would be interesting at the theoretical and practical levels, as we could then improve our prototype by extending it with those new features.

Similarly, adding higher order capabilities by an extension of $CRWL^\sigma_{\pi^\alpha}$ in the line of [21], and implementing them by means of the classic transformation of [53], would also be interesting and it is standard in the field of FLP. Then, for example, we could define a more generic version of `discoverHow` with an additional argument for the function used to perform a deduction step (`discStepHow` in our dungeon problem). This higher order version of $CRWL^\sigma_{\pi^\alpha}$ could also be used to face the challenges regarding the implementation of type classes in FLP through the classical transformational technique of [52] pointed out by Lux in [39]. Although some solutions based on the frameworks of [35, 32] were already proposed in [46] we think that an alternative based on $CRWL^\sigma_{\pi^\alpha}$ would be better thanks to its clean and compositional semantics. More novel would be using the matching-modulo capacities of Maude to enhance the expressiveness of the semantics, after a corresponding revision of the theory of $CRWL^\sigma_{\pi^\alpha}$. Besides, as mentioned at the end of Section 5.5, some additional research must be done to improve the performance of the interpreter. As we pointed out there, an explicit representation of non-determinism on a deterministic language seems promising [5, 12, 11, 19]. Some possible concretization of this idea could be using Maude functional modules with `strat` annotations using the techniques in [37], or adapting the transformation in [10].

As suggested in Section 5.4, finding a criterion for the equivalence of plurality maps and defining more equational laws for non-determinism, besides the restricted forms of bubbling proposed here, would improve the understanding of programs, which could finally lead to the development of more interesting

program examples that maybe could illustrate the interest of the semantics. In this line we also find interesting the relation between the different notions of determinism entailed by *CRWL*, π^{α}*CRWL* and π^{β}*CRWL*, and the relation between confluence of term rewriting and those notions of determinism. We already made some advances in this line in previous works [33, 30].

To conclude, an investigation of the technical relation between π^{β}*CRWL* and the plural semantics from [9] which inspired it, would be very interesting. We conjecture a strong semantic equivalence between them.

References

[1] E. Albert, M. Hanus, F. Huch, J. Oliver, and G. Vidal. Operational semantics for declarative multi-paradigm languages. *Journal of Symbolic Computation*, 40(1):795–829, 2005.

[2] S. Antoy, D. Brown, and S. Chiang. Lazy context cloning for non-deterministic graph rewriting. In *Proc. Termgraph'06,*, pages 61–70. ENTCS, 176(1), 2007.

[3] S. Antoy and M. Hanus. Functional logic design patterns. In Z. Hu and M. Rodríguez-Artalejo, editors, *FLOPS*, volume 2441 of *Lecture Notes in Computer Science*, pages 67–87. Springer, 2002.

[4] S. Antoy and M. Hanus. Functional logic programming. *Communications of the ACM*, 53(4):74–85, 2010.

[5] S. Antoy, P. J. Iranzo, and B. Massey. Improving the efficiency of non-deterministic computations. *Electr. Notes Theor. Comput. Sci.*, 64:73–94, 2002.

[6] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *Proceedings of 22nd Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL 1995*, pages 233–246. ACM, 1995.

[7] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[8] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Proceedings Second International Workshop on Rewriting Logic and its Applications, WRLA'98, Pont-à-Mousson, France, September 1–4, 1998*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 329–344. Elsevier, 1998. `http://www.elsevier.nl/locate/entcs/volume15.html`.

[9] B. Braßel and R. Berghammer. Functional (logic) programs as equations over order-sorted algebras. In *Informal Proceedings 19th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR 2009*, 2009.

[10] B. Braßel, S. Fischer, M. Hanus, and F. Reck. Transforming functional logic programs into monadic functional programs. In J. Mariño, editor, *WFLP*, volume 6559 of *Lecture Notes in Computer Science*, pages 30–47. Springer, 2010.

[11] B. Braßel, M. Hanus, B. Peemöller, and F. Reck. Kics2: A new compiler from curry to haskell. In H. Kuchen, editor, *WFLP*, volume 6816 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[12] B. Braßel and F. Huch. On a tighter integration of functional and logic programming. In Z. Shao, editor, *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 2007.

[13] R. Caballero and F. López-Fraguas. Improving deterministic computations in lazy functional logic languages. *Journal of Functional and Logic Programming*, 2003(1), 2003.

[14] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[15] D. DeGroot and G. e. Lindstrom. *Logic Programming, Functions, Relations, and Equations.* Prentice Hall, 1986.

[16] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[17] R. Echahed and J.-C. Janodet. Admissible Graph Rewriting and Narrowing. In *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming*, pages 325 – 340, Manchester, June 1998. MIT Press.

[18] S. Escobar. Implementing natural rewriting and narrowing efficiently. In Y. Kameyama and P. Stuckey, editors, *Proceedings of the 7th International Symposium on Functional and Logic Programming, FLOPS 2004*, volume 2998 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2004.

[19] S. Fischer, O. Kiselyov, and C.-c. Shan. Purely functional lazy non-deterministic programming. In *ICFP '09: Proceedings of the 14th ACM SIGPLAN international conference on Functional programming*, pages 11–22, New York, NY, USA, 2009. ACM.

[20] K. Futatsugi and R. Diaconescu. *CafeOBJ Report.* World Scientific, AMAST Series, 1998.

[21] J. González-Moreno, M. Hortalá-González, and M. Rodríguez-Artalejo. A higher order rewriting logic for functional logic programming. In *Proc. International Conference on Logic Programming, ICLP 1997*, pages 153–167. MIT Press, 1997.

[22] J. C. González-Moreno, T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. A Rewriting Logic for Declarative Programming. In *Proc. European Symposium on Programming, ESOP 1996*, volume 1058 of *Lecture Notes in Computer Science*, pages 156–172. Springer, 1996.

[23] J. C. González-Moreno, T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *Journal of Logic Programming*, 40(1):47–87, 1999.

[24] M. Hanus. Functional logic programming: From theory to Curry. Technical report, Christian-Albrechts-Universität Kiel, 2005.

[25] M. Hanus. Multi-paradigm declarative languages. In *Proceedings of the International Conference on Logic Programming, ICLP 2007*, pages 45–75. Springer LNCS 4670, 2007.

[26] 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.

[27] J. Hughes and J. O'Donnell. Expressing and Reasoning About Non-deterministic Functional Programs. In *Proceedings of the 1989 Glasgow Workshop on Functional Programming*, Workshops in Computing, pages 308–328, London, UK, 1990. Springer.

[28] H. Hussmann. *Non-Determinism in Algebraic Specifications and Algebraic Programs*. Birkhäuser Verlag, 1993.

[29] J. Launchbury. A natural semantics for lazy evaluation. In *Proc. ACM Symposium on Principles of Programming Languages, POPL 1993*, pages 144–154. ACM Press, 1993.

[30] F. López-Fraguas, E. Martin-Martin, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. Rewriting and narrowing for constructor systems with call-time choice semantics. *Submitted to Theory and Practice of Logic Programming, available under request*, 2010.

[31] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A Fully Abstract Semantics for Constructor Systems. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications, RTA 2009*, volume 5595 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2009.

[32] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A lightweight combination of semantics for non-deterministic functions. *CoRR*, abs/0903.2205, 2009.

[33] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A simple rewrite notion for call-time choice semantics. In *Proc. Principles and Practice of Declarative Programming*, pages 197–208. ACM Press, 2007.

[34] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. Rewriting and call-time choice: the HO case. In *Proc. 9th International Symposium on Functional and Logic Programming, FLOPS 2008*, volume 4989 of *LNCS*, pages 147–162. Springer, 2008.

[35] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A flexible framework for programming with non-deterministic functions. In *PEPM '09: Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 91–100. ACM, 2009.

[36] F. López-Fraguas and J. Sánchez-Hernández. $\mathcal{TOY}$: A multiparadigm declarative system. In *Proc. Rewriting Techniques and Applications, RTA 1999*, pages 244–247. Springer LNCS 1631, 1999.

[37] S. Lucas. Needed reductions with context-sensitive rewriting. In M. Hanus, J. Heering, and K. Meinke, editors, *ALP/HOA*, volume 1298 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 1997.

[38] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1), 1998.

[39] W. Lux. Curry mailing list: Type-classes and call-time choice vs. run-time choice. `http://www.informatik.uni-kiel.de/~curry/listarchive/0790.html`, August 2009.

[40] J. McCarthy. A Basis for a Mathematical Theory of Computation. In *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, 1963.

[41] R. J. Plasmeijer and M. C. J. D. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.

[42] D. Plump. *Term Graph Rewriting*, volume 2: Applications, Languages, and Tools of *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 3–61. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[43] A. Riesco and J. Rodríguez-Hortalá. A natural implementation of plural semantics in Maude. In T. Ekman and J. Vinju, editors, *Proceedings of the 9th Workshop on Language Descriptions, Tools, and Applications, LDTA 2009*, volume 253(7) of *Electronic Notes in Computer Science*, pages 165–175. Elsevier, 2010.

[44] A. Riesco and J. Rodríguez-Hortalá. Programming with singular and plural non-deterministic functions. In *PEPM '10: Proceedings of the 2010 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 83–92. ACM, 2010.

[45] J. Rodríguez-Hortalá. A hierarchy of semantics for non-deterministic term rewriting systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, Leibniz International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008.

[46] J. Rodríguez-Hortalá. Curry mailing list: Re: Type-classes and call-time choice vs. run-time choice. `http://www.informatik.uni-kiel.de/~curry/listarchive/0801.html`, August 2009.

[47] H. Søndergaard and P. Sestoft. Referential transparency, definiteness and unfoldability. *Acta Inf.*, 27(6):505–517, 1990.

[48] H. Søndergaard and P. Sestoft. Non-determinism in functional languages. *The Computer Journal*, 35(5):514–523, 1992.

[49] L. Sterling and E. Shapiro. *The Art of Prolog*. MIT Press, 1986.

[50] TeReSe. *Term Rewriting Systems*, volume No. 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[51] P. Wadler. How to replace failure by a list of successes. In *Proc. Functional Programming and Computer Architecture*. Springer LNCS 201, 1985.

[52] P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad hoc. In *Pro.16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 60–76, New York, NY, USA, 1989. ACM.

[53] D. H. Warren. Higher-order extensions to Prolog: are they needed? In J. Hayes, D. Michie, and Y.-H. Pao, editors, *Machine Intelligence 10*, pages 441–454. Ellis Horwood Ltd., 1982.

A Additional proofs for the results

When performing proofs by induction we will usually use IH to refer to the induction hypothesis of the current induction. We will also use the following auxiliary result.

Lemma 9. *For any program, $t, t' \in CTerm_\bot$, $e \in Exp_\bot$, $\sigma, \sigma' \in Subst_\bot$*

1. *We have that $\mathcal{P} \vdash_{\pi^\alpha CRWL} t \twoheadrightarrow t'$ iff $t' \sqsubseteq t$, and also $\mathcal{P} \vdash_{\pi^\beta CRWL} t \twoheadrightarrow t'$ iff $t' \sqsubseteq t$.*
2. *If $\sigma \sqsubseteq \sigma'$ then $e\sigma \sqsubseteq e\sigma'$.*

Proof. 1. By a simple induction on the structure of t

2. A simple induction on the structure of e.

□

A.1 For Section 3

Proof for Lemma 1. We will prove both inclusions. First, to prove $dom(?\{\theta_1 \dots \theta_n\}) \subseteq \bigcup_i dom(\theta_i)$ consider some $X \notin \bigcup_i dom(\theta_i)$, then $\rho_1 \dots \rho_m \equiv []$ and $?\{\theta_1 \dots \theta_n\}(X) \equiv X$, hence $X \notin dom(?\{\theta_1 \dots \theta_n\})$. Now, to prove $\bigcup_i dom(\theta_i) \subseteq dom(?\{\theta_1 \dots \theta_n\})$, let us consider some $X \in \bigcup_i dom(\theta_i)$. Then $\exists \theta_i \in \{\theta_1, \dots, \theta_n\}$ such that $X \in dom(\theta_i)$. But that implies $\rho_1 \dots \rho_m \not\equiv []$, thus $X \; ? \; \rho_1(X) \; ? \; \dots \; ? \; \rho_m(X) \not\equiv X$ and so $?\{\theta_1 \dots \theta_n\}(X) \not\equiv X$, because $\theta_1(X) \; ? \; \dots \; ? \; \theta_n(X) \not\equiv X$ anyway. Therefore $X \in dom(?\{\theta_1 \dots \theta_n\})$. □

Theorem 1 is based upon the following facts concerning the conmutativity, associativity of ?, and the idempotence of the partial application of ?.

Lemma 10. *For any program, $\mathcal{C} \in Cntxt$ and $e_1, e_2, e_3 \in Exp_\bot$:*

1. $[\![\mathcal{C}[e_1 \; ? \; e_2]]\!]^{\alpha pl} = [\![\mathcal{C}[e_2 \; ? \; e_1]]\!]^{\alpha pl}$
2. $[\![\mathcal{C}[(e_1 \; ? \; e_2) \; ? \; e_3]]\!]^{\alpha pl} = [\![\mathcal{C}[e_1 \; ? \; (e_2 \; ? \; e_3)]]\!]^{\alpha pl}$
3. $[\![\mathcal{C}[e_1 \; ? \; e_1]]\!]^{\alpha pl} = [\![\mathcal{C}[e_1]]\!]^{\alpha pl}$
4. $[\![\mathcal{C}[e_1]]\!]^{\alpha pl} \subseteq [\![\mathcal{C}[e_1 \; ? \; e_2]]\!]^{\alpha pl}$. *As a consequence, for any pair of finite chains $a_1 \dots a_n \in Exp_\bot^*$, $b_1 \dots b_m \in Exp_\bot^*$ if $\{a_1, \dots, a_n\} \subseteq \{b_1, \dots, b_m\}$ then for any context $\mathcal{C}$ $[\![\mathcal{C}[a_1 \; ? \; \dots \; ? \; a_n]]\!]^{\alpha pl} \subseteq [\![\mathcal{C}[b_1 \; ? \; \dots \; ? \; b_m]]\!]^{\alpha pl}$.*

Proof. 1. We have to prove that for any $t \in CTerm_\perp$ if $\mathcal{C}[e_1 \ ? \ e_2] \twoheadrightarrow t$ then $\mathcal{C}[e_2 \ ? \ e_1] \twoheadrightarrow t$ and vice versa. This can be easily done with a simple induction on the size of the proof which acts as hypothesis.

2. Similar to the previous item.

3. Similar to the previous item.

4. Assume $\mathcal{C}[e_1] \twoheadrightarrow t$, we can prove that then $\mathcal{C}[e_1 \ ? \ e_2] \twoheadrightarrow t$ with a simple induction on the size of the proof for $\mathcal{C}[e_1] \twoheadrightarrow t$. Regarding the second part of this item, assume $\mathcal{C}[a_1 \ ? \ \ldots \ ? \ a_n] \twoheadrightarrow t$, then by the previuos items we may eliminate repeated elements in the chains and arrange them in way such that $b_1 \ldots b_m \equiv a_1 \ldots a_n b'_1 \ldots b'_k$ with $n + k = m$. Then by the first part of this item

$$\begin{array}{l}[\![\mathcal{C}[a_1 \ ? \ \ldots \ ? \ a_n]]\!]^{\alpha pl} \subseteq [\![\mathcal{C}[a_1 \ ? \ \ldots \ ? \ a_n \ ? \ b'_1]]\!]^{\alpha pl} \\ \subseteq \ldots \subseteq [\![\mathcal{C}[a_1 \ ? \ \ldots \ ? \ a_n \ ? \ b'_1 \ ? \ \ldots \ ? \ b'_k]]\!]^{\alpha pl} = [\![\mathcal{C}[b_1 \ ? \ \ldots \ ? \ b_m]]\!]^{\alpha pl}\end{array}$$

and this process ends because both chains are finite.

□

Proof for Theorem 1. First we will prove that for any $t \in CTerm_\perp$, if $\mathcal{C}[e] \twoheadrightarrow t$ then $\exists\{s_1, \ldots, s_n\} \subseteq [\![e]\!]^{\alpha pl}$ such that $\mathcal{C}[s_1 \ ? \ \ldots \ ? \ s_n] \twoheadrightarrow t$, we proceed by induction on the size K of the proof for $\mathcal{C}[e] \twoheadrightarrow t$.

Base cases $K = 1$:

B Then we can take $\{s_1, \ldots, s_n\} = \{\perp\}$ to do $\mathcal{C}[\perp] \twoheadrightarrow \perp$, by **B**.

RR, DC These cases correspond to $X \twoheadrightarrow X$ by **RR** and $c \twoheadrightarrow c$ by **DC**. Then $\mathcal{C} = []$ and so the hypothesis was $e \equiv \mathcal{C}[e] \twoheadrightarrow t$. Hence we can take $\{s_1, \ldots, s_n\} = \{t\}$ to do $\mathcal{C}[s_1 \ ? \ \ldots \ ? \ s_n] \equiv [t] \equiv t \twoheadrightarrow t$, by Lemma 9.

Inductive steps $K > 1$:

DC If $\mathcal{C} = []$ then we are done like in the previous step. Otherwise we have:

$$\frac{e_1 \twoheadrightarrow t_1 \quad \ldots \quad \mathcal{C}'[e] \twoheadrightarrow t' \quad \ldots \quad e_l \twoheadrightarrow t_l}{\mathcal{C}[e] \equiv c(e_1, \ldots, \mathcal{C}'[e], \ldots, e_l) \twoheadrightarrow c(t_1, \ldots, t', \ldots, t_l) \equiv t} \ \mathbf{DC}$$

Then by IH $\exists\{s_1, \ldots, s_n\} \subseteq [\![e]\!]^{\alpha pl}$ such that $\mathcal{C}'[s_1 \ ? \ \ldots \ ? \ s_n] \twoheadrightarrow t'$, therefore we can build the following proof:

$$\frac{e_1 \twoheadrightarrow t_1 \quad \ldots \quad \mathcal{C}'[s_1 \ ? \ \ldots \ ? \ s_n] \twoheadrightarrow t' \quad \ldots \quad e_l \twoheadrightarrow t_l}{\mathcal{C}[s_1 \ ? \ \ldots \ ? \ s_n] \equiv c(e_1, \ldots, \mathcal{C}'[s_1 \ ? \ \ldots \ ? \ s_n], \ldots, e_l) \twoheadrightarrow c(t_1, \ldots, t', \ldots, t_l) \equiv t} \ \mathbf{DC}$$

POR$^\alpha$ If $\mathcal{C} = []$ then we are done like in the previous step. Otherwise we have:

$$\frac{\begin{array}{ccccccc} e_1 \twoheadrightarrow p_1\theta_{11} & & \mathcal{C}'[e] \twoheadrightarrow p'\theta'_1 & & e_l \twoheadrightarrow p_l\theta_{l1} & & \\ \ldots & \ldots & \ldots & \ldots & \ldots & & \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & \mathcal{C}'[e] \twoheadrightarrow p'\theta'_{m'} & & e_l \twoheadrightarrow p_l\theta_{lm_l} & & r\theta \twoheadrightarrow t \end{array}}{\mathcal{C}[e] \equiv f(e_1, \ldots, \mathcal{C}'[e], \ldots, e_l) \twoheadrightarrow t} \ \mathbf{POR}^\alpha$$

with $\theta = ?(\theta_{11} \ldots \theta_{1m_1}) \uplus \ldots \uplus ?(\theta'_1 \ldots \theta'_{m'}) \uplus \ldots \uplus ?(\theta_{l1} \ldots \theta_{lm_l}) \uplus \theta_e$ for some $(f(p_1, \ldots, p', \ldots, p_l) \to r) \in \mathcal{P}$. Then by IH for each $\theta'_i \in \{\theta'_1, \ldots, \theta'_{m'}\}$ $\exists\{s_{i1}, \ldots, s_{in_i}\} \subseteq [\![e]\!]^{\alpha pl}$ such that $\mathcal{C}'[s_{i1} \ ? \ \ldots \ ? \ s_{in_i}] \twoheadrightarrow p'\theta'_i$, hence $\mathcal{C}'[s_{11} \ ? \ \ldots \ ? \ s_{1n_1} \ ? \ \ldots \ ? \ s_{m'1} \ ? \ \ldots \ ? \ s_{m'n_{m'}}] \twoheadrightarrow p'\theta'_i$ by Lemma 10. Therefore we can take:

$$\begin{array}{c}\{s_1, \ldots, s_n\} = \{s_{11} \ ? \ \ldots \ ? \ s_{1n_1} \ ? \ \ldots \ ? \ s_{m'1} \ ? \ \ldots \ ? \ s_{m'n_{m'}}\} \\ a \equiv \mathcal{C}'[s_{11} \ ? \ \ldots \ ? \ s_{1n_1} \ ? \ \ldots \ ? \ s_{m'1} \ ? \ \ldots \ ? \ s_{m'n_{m'}}]\end{array}$$

to build the following proof:

$$\frac{\begin{array}{ccccccc} e_1 \twoheadrightarrow p_1\theta_{11} & & a \twoheadrightarrow p'\theta'_1 & & e_l \twoheadrightarrow p_l\theta_{l1} & & \\ \ldots & \ldots & \ldots & \ldots & \ldots & & \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & a \twoheadrightarrow p'\theta'_{m'} & & e_l \twoheadrightarrow p_l\theta_{lm_l} & & r\theta \twoheadrightarrow t \end{array}}{\mathcal{C}[s_1 \ ? \ \ldots \ ? \ s_n] \equiv f(e_1, \ldots, a, \ldots, e_l) \twoheadrightarrow t} \ \mathbf{POR}^\alpha$$

Now we have to prove the other implication, that is, given $\{s_1, \ldots, s_n\} \subseteq [\![e]\!]^{\alpha pl}$ such that $\mathcal{C}[s_1 \ ? \ \ldots \ ? \ s_n] \twoheadrightarrow t$ then $\mathcal{C}[e] \twoheadrightarrow t$. If $\mathcal{C} = []$ then the hypothesis is $s_1 \ ? \ \ldots \ ? \ s_n \twoheadrightarrow t$, so it must exists some $s_i \in \{s_1, \ldots, s_n\}$ such that $s_i \twoheadrightarrow t$. But then $t \sqsubseteq s_i$ by Lemma 9, as $s_i \in CTerm_\bot$, as it is a value for e. But then the hypothesis $e \twoheadrightarrow s_i$ and $t \sqsubseteq s_i$ implies $e \twoheadrightarrow t$ by the monotonicity of Proposition 1.
To prove the case when $\mathcal{C} \neq []$ we need to do a simple induction on the size of $\mathcal{C}[s_1 \ ? \ \ldots \ ? \ s_n] \twoheadrightarrow t$, in which we will not assume $\mathcal{C} \neq []$. □

Proof for Proposition 1. By induction on the structure of $e \twoheadrightarrow t$.

Base cases

B $e \twoheadrightarrow \bot \equiv t$. Then $t' \sqsubseteq t$ implies $t' \equiv \bot$, so $e' \twoheadrightarrow \bot \equiv t'$, by **B**.

RR $e \equiv X \twoheadrightarrow X \equiv t$. Then $t' \sqsubseteq t$ implies $t' \equiv \bot$ or $t' \equiv X$. In the first case we proceed like in the case for **B**, in the latter as $X \equiv e \sqsubseteq e'$ implies $e' \equiv X$, so $e' \equiv X \twoheadrightarrow X \equiv t'$ by **RR**.

DC $e \equiv c \twoheadrightarrow c \equiv t$. Very similar to the case for **RR**.

Inductive steps

DC Then we have

$$\frac{e_1 \twoheadrightarrow t_1 \quad \ldots \quad e_n \twoheadrightarrow t_n}{e \equiv c(e_1, \ldots, e_n) \twoheadrightarrow c(t_1, \ldots, t_n) \equiv t} \ \mathbf{DC}$$

Then $t' \sqsubseteq t$ implies $t' \equiv \bot$ or $t' \equiv c(t'_1, \ldots, t'_n)$ with $t'_i \sqsubseteq t_i$ for every $i \in \{1, \ldots, n\}$. In the first case we proceed like in the case for **B**, in the latter as $c(e_1, \ldots, e_n) \equiv e \sqsubseteq e'$ implies $e' \equiv c(e'_1, \ldots, e'_n)$ with $e_i \sqsubseteq e'_i$ for every $i \in \{1, \ldots, n\}$ then by IH we get $e'_i \twoheadrightarrow t'_i$ for every $i \in \{1, \ldots, n\}$, and we can build the following proof:

$$\frac{e'_1 \twoheadrightarrow t_1 \quad \ldots \quad e'_n \twoheadrightarrow t_n}{e' \equiv c(e'_1, \ldots, e'_n) \twoheadrightarrow c(t'_1, \ldots, t'_n) \equiv t'} \ \mathbf{DC}$$

POR$^\alpha$ Then we have

$$\frac{\begin{array}{ccccc} e_1 \twoheadrightarrow p_1\theta_{11} & & & & e_n \twoheadrightarrow p_n\theta_{n1} \\ \ldots & & \ldots & & \ldots \\ e_1 \twoheadrightarrow p_1\theta_{1m_1} & & & & e_n \twoheadrightarrow p_n\theta_{nm_n} \qquad r\theta \twoheadrightarrow t \end{array}}{e \equiv f(e_1, \ldots, e_n) \twoheadrightarrow t} \ \mathbf{POR}^\alpha$$

with $\theta = ?(\theta_{11} \ldots \theta_{1m_1}) \uplus \ldots \uplus ?(\theta_{n1} \ldots \theta_{nm_n}) \uplus \theta_e$, for some $(f(p_1, \ldots, p_n) \to r) \in \mathcal{P}$. Then as $f(e_1, \ldots, e_n) \equiv e \sqsubseteq e'$ implies $e' \equiv f(e'_1, \ldots, e'_n)$ with $e_i \sqsubseteq e'_i$ for every $i \in \{1, \ldots, n\}$ then by IH $\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m_i\}$ $e'_i \twoheadrightarrow p_i\theta_{ij}$. We can also apply the IH to get $r\theta \twoheadrightarrow t'$, as $t' \sqsubseteq t$ by hypothesis, and build the following proof:

$$\frac{\begin{array}{ccccc} e'_1 \twoheadrightarrow p_1\theta_{11} & & & & e'_n \twoheadrightarrow p_n\theta_{n1} \\ \ldots & & \ldots & & \ldots \\ e'_1 \twoheadrightarrow p_1\theta_{1m_1} & & & & e'_n \twoheadrightarrow p_n\theta_{nm_n} \qquad r\theta \twoheadrightarrow t' \end{array}}{e' \equiv f(e'_1, \ldots, e'_n) \twoheadrightarrow t'} \ \mathbf{POR}^\alpha$$

□

Some of the following interesting facts about the preorders $\sqsubseteq_\pi$ and $\trianglelefteq$ will be used in the proof for Proposition 2.

Proposition 6. *1. $\forall \theta, \theta' \in CSubst_\bot$, $\theta \sqsubseteq \theta'$ iff $\theta \sqsubseteq_\pi \theta'$.*

2. $\sqsubseteq_\pi \subseteq CSubst^?_\bot \times CSubst^?_\bot$ is a preorder but not a partial order.

3. Given $\theta, \theta' \in CSubst^?_\bot$ if $\theta \sqsubseteq_\pi \theta'$ then $\theta \trianglelefteq \theta'$

4. $\trianglelefteq \subseteq Subst_\bot \times Subst_\bot$ is a preorder but not a partial order

Proof. 1. By definition.

2. It is very easy to check that it is reflexive and transitive, but it is not antisymetric, as $[X/0] \sqsubseteq_\pi [X/0 \ ? \ 0]$, $[X/0 \ ? \ 0] \sqsubseteq_\pi [X/0]$ but $[X/0] \neq [X/0 \ ? \ 0]$.

3. For any $X \in \mathcal{V}$, if $\theta(X) = t_1 \ ? \ \ldots \ ? \ t_n$ and $\theta'(X) = t'_1 \ ? \ \ldots \ ? \ t'_m$ (note that $\theta(X)$ has that shape even when $X \notin dom(\theta)$, as X has that shape) and $\theta(X) \twoheadrightarrow t$ then it must exist some $t_i \in \{t_1, \ldots, t_n\}$ such that $t_i \twoheadrightarrow t$, and so $t \sqsubseteq t_i$ by Lemma 9. But then $t \sqsubseteq t_i \sqsubseteq t'_j$ for some $t_j \in \{t'_1, \ldots, t'_m\}$, because $\theta \sqsubseteq_\pi \theta'$, hence $t'_j \twoheadrightarrow t$ by Lemma 9 and so $\theta'(X) \twoheadrightarrow t$.

4. It is very easy to check that it is reflexive and transitive, but it is not antisymetric, because given $\mathcal{P} = \{f \to 1, g \to 1\}$ we have $[X/f] \trianglelefteq [X/g]$ and $[X/g] \trianglelefteq [X/f]$ while $[X/f] \neq [X/g]$.

□

The following standard notions from term rewriting [7] will also be used in the proof for Proposition 2.

Definition 8. • *The set of positions of an expression e is the set $\mathcal{O}e$ of strings over the alfabet of positive integers defined as $\mathcal{O}X = \{\epsilon\}$, if $X \in \mathcal{V}$; $\mathcal{O}h(e_1, \ldots, e_n) = \{\epsilon\} \cup \bigcup_{i \in \{1,\ldots,n\}} \{i.p \mid p \in \mathcal{O}e_i\}$ otherwise, where ϵ denotes the empty string and $\_.\_$ is used for concatenation.*
We say that two positions are parallel if none of them is prefix of the other.

- *For any $e \in Exp_\perp$, $p \in \mathcal{O}e$, the subexpression of e at position p denoted by $s|_p$, is defined as $e|_\epsilon = e$; $h(e_1, \ldots, e_n)|_{i.q} = e_i|_q$.*

- *For any $e, e' \in Exp_\perp$, $p \in \mathcal{O}e$, by $e[e']_p$ we denote the expression obtained from e by replacing the subexpression at position p by e', defined as $e[e']_\epsilon = e'$; $f(e_1, \ldots, e_n)[e']_{i.q} = f(e_1, \ldots, e_i[e']_q, \ldots, e_n)$.*

- *As one-hole context can be understood as funcions $\mathcal{C} : Exp_\perp \to Exp_\perp$, for any $\mathcal{C} \in Cntx$ we may assume some $e \in Exp_\perp, p \in \mathcal{O}e$ such that $\mathcal{C} = \lambda e'.e[e']_p$. With this in mind Theorem 1 can be recasted as $[\![e[e']_p]\!]^{\alpha pl} = \bigcup_{\overline{t} \subseteq [\![e']\!]^{\alpha pl}} [\![e[?\overline{t}]_p]\!]^{\alpha pl}$, where $?\{t_1, \ldots, t_n\}$ denotes $t_1 \ ? \ \ldots \ ? \ t_n$ for some arrangement of the elements of $\{t_1, \ldots, t_n\}$ in $t_1 \ ? \ \ldots \ ? \ t_n$.*

Proof for Proposition 2. 1. If $e \equiv X \in \mathcal{V}$, assume $e\sigma \equiv \sigma(X) \twoheadrightarrow t$, then $e\sigma' \equiv \sigma'(X) \twoheadrightarrow t$ with a proof of the same size or smaller, by hypothesis. Otherwise we proceed by induction on the structure of $e\sigma \twoheadrightarrow t$.

Base cases

B Then $t \equiv \perp$ and $e\sigma' \twoheadrightarrow \perp$ with a proof of size 1 just applying rule **B**.

RR Then $e \in \mathcal{V}$ and we are in the previous case.

DC Then $e \equiv c \in CS^0$, as $e \notin \mathcal{V}$, hence $e\sigma \equiv c \equiv e\sigma'$ and every proof for $e\sigma \twoheadrightarrow t$ is a proof for $e\sigma' \twoheadrightarrow t$.

Inductive steps

DC Then $e \equiv c(e_1, \ldots, e_n)$, as $e \notin \mathcal{V}$, and we have:

$$\frac{e_1\sigma \twoheadrightarrow t_1 \quad \ldots \quad e_n\sigma \twoheadrightarrow t_n}{e\sigma \equiv c(e_1\sigma, \ldots, e_n\sigma) \twoheadrightarrow c(t_1, \ldots, t_n) \equiv t} \ \mathbf{DC}$$

By IH or the proof of the other cases $\forall i \in \{1, \ldots, n\}$ we have $e_i\sigma' \twoheadrightarrow t_i$ with a proof of the same size or smaller, so we can built a proof for $e\sigma' \equiv c(e_1\sigma', \ldots, e_n\sigma') \twoheadrightarrow c(t_1, \ldots, t_n) \equiv t$ using **DC**, with a size equal or smaller than the size of the starting proof.

POR$^\alpha$ Very similar to the proof of the previuos case. We have $e \equiv f(e_1, \ldots, e_n)$ (as $e \notin \mathcal{V}$) and given a proof for $e\sigma \equiv f(e_1, \ldots, e_n)\sigma \twoheadrightarrow t$, we apply the IH to every $e_i\sigma \twoheadrightarrow p_i\theta_{ij}$ to get that $e_i\sigma' \twoheadrightarrow p_i\theta_{ij}$ with a proof of the same size or smaller. But then we can use these proofs in a **POR**$^\alpha$ step from $e\sigma' \equiv f(e_1\sigma', \ldots, e_n\sigma')$ and use the same substitution $\theta \in CSubst^?_\perp$ for parameter passing, constructing a proof with a size equal or smaller than the size of the starting one.

2. If $\theta \sqsubseteq \theta'$ then for any $X \in \mathcal{V}$ we have $\theta(X) \sqsubseteq \theta'(X)$, hence if $\theta(X) \twoheadrightarrow t$ then $\theta'(X) \twoheadrightarrow t$ with a proof of the same size or smaller, by the polarity of $\pi^\alpha CRWL$ from Proposition 1. But then we can apply the strong monotonicity of $Subst_\perp$ from the previous item, to get the desired result.

3. Using the notations of Definition 8, given $X_i \in \overline{X} = var(e)$ if the set of positions of the occurrences of X_i in e is $\{p_{i1}, \ldots, p_{im_i}\}$ then $e \equiv e[X_i]_{p_{i1}} \equiv (e[X_i]_{p_{i1}})[X_i]_{p_{i2}} \equiv \ldots e[X_i]_{p_{i1}} \ldots [X_i]_{p_{im_i}}$. As the positions of any pair of different occurrences of (possibly different) variables are parallel, we can do this for every variable in $\overline{X}$ to get $e \equiv e[Y_1]_{o_1} \ldots [Y_m]_{o_m}$, where $\{o_1, \ldots, o_m\}$ is the set of positions

of every occurrence in e of any variable in $var(e)$ and $\{Y_1, \ldots, Y_m\} = \overline{X}$. Note how each position in $\{o_1, \ldots, o_m\}$ is parallel to each other. But then we can apply the recasted version of Theorem 1 that appears in Definition 8, to get:

$$\begin{aligned}
&[\![e\sigma]\!]^{\alpha pl} = [\![e[Y_1\sigma]_{o_1} \ldots [Y_m\sigma]_{o_m}]\!]^{\alpha pl} \\
&= \bigcup_{\overline{t_1} \subseteq [\![Y_1\sigma]\!]^{\alpha pl}} [\![e[?\overline{t_1}]_{o_1} \ldots [Y_m\sigma]_{o_m}]\!]^{\alpha pl} && \text{by Theorem 1} \\
&= \bigcup_{\overline{\overline{t}} \subseteq [\![Y\sigma]\!]^{\alpha pl}} [\![e[?\overline{\overline{t}}]_{\overline{o}}]\!]^{\alpha pl} && \text{by Theorem 1 (many times)} \\
&= \bigcup_{\overline{\overline{t}} \subseteq [\![Y\sigma']\!]^{\alpha pl}} [\![e[?\overline{\overline{t}}]_{\overline{o}}]\!]^{\alpha pl} && \text{as } \sigma \trianglelefteq \sigma' \\
&= [\![e[Y_1\sigma']_{o_1} \ldots [Y_m\sigma']_{o_m}]\!]^{\alpha pl} = [\![e\sigma']\!]^{\alpha pl} && \text{by Theorem 1}
\end{aligned}$$

Note that we cannot claim $e\sigma' \twoheadrightarrow t$ with proof of the same size of smaller, as we can see for example with $\sigma = [X/0] \trianglelefteq [X/0\ ?\ 0] = \sigma', e \equiv X, t \equiv 0$ for which $e\sigma \equiv 0 \twoheadrightarrow 0$ with size one but $e\sigma' \equiv 0\ ?\ 0 \twoheadrightarrow 0$ with size greater or equal to four.

4. If $\theta \sqsubseteq_\pi \theta'$ then $\theta \trianglelefteq \theta'$ by Proposition 6, hence this item holds by the previous item.

□

For Proposition 3. For the left to right inclusion we must prove that given $\vdash_{\pi^\alpha CRWL} c\mathcal{C}[e] \twoheadrightarrow t$ then $\exists s \in [\![e]\!]^{\alpha pl}$ such that $\vdash_{\pi^\alpha CRWL} c\mathcal{C}[s] \twoheadrightarrow t$. We proceed by induction on the size of the $\pi^\alpha CRWL$-proof for $\vdash_{\pi^\alpha CRWL} c\mathcal{C}[e] \twoheadrightarrow t$. Concerning the base cases:

B Then $t \equiv \bot$ and we can take $s \equiv \bot$ for which $e \twoheadrightarrow \bot$ and $c\mathcal{C}[s] \twoheadrightarrow \bot \equiv t$ by rule **B** in both cases.

RR, DC Then either $c\mathcal{C}[e] \equiv X \in \mathcal{V}$ or $c\mathcal{C}[e] \equiv c \in CS^0$, therefore $c\mathcal{C} = []$ and thus $e \equiv c\mathcal{C}[e] \twoheadrightarrow t$ by hypothesis. But then we can take $s \equiv t$ so we have $c\mathcal{C}[s] \equiv t \twoheadrightarrow t$ by Lemma 9 (page 36).

Regarding the inductive step:

DC If $c\mathcal{C} = []$ we proceed like in the previous case, otherwise we have:

$$\frac{e_1 \twoheadrightarrow t_1 \quad \ldots \quad c\mathcal{C}'[e] \twoheadrightarrow t' \quad \ldots \quad e_n \twoheadrightarrow t_n}{c\mathcal{C}[e] \equiv c(e_1, \ldots, c\mathcal{C}'[e], \ldots, e_n) \twoheadrightarrow c(t_1, \ldots, t', \ldots, t_n) \equiv t}\ \mathbf{DC}$$

But then we can apply the IH to $c\mathcal{C}'[e] \twoheadrightarrow t'$ to get some $s \in [\![e]\!]^{\alpha pl}$ such that $c\mathcal{C}'[s] \twoheadrightarrow t'$, so we can build the following proof:

$$\frac{\dfrac{}{e_1 \twoheadrightarrow t_1}\ hypothesis \quad \ldots \quad \dfrac{}{c\mathcal{C}'[s] \twoheadrightarrow t'}\ IH \quad \ldots \quad \dfrac{}{e_n \twoheadrightarrow t_n}\ hypothesis}{c\mathcal{C}[s] \equiv c(e_1, \ldots, c\mathcal{C}'[s], \ldots, e_n) \twoheadrightarrow c(t_1, \ldots, t', \ldots, t_n) \equiv t}\ \mathbf{DC}$$

POR$^\alpha$ If $c\mathcal{C} = []$ we proceed like in the previous case, otherwise we have $c\mathcal{C} = f(e_1, \ldots, c\mathcal{C}', \ldots, e_n)$ for some $f \in FS$, which contradicts the hypothesis as $c\mathcal{C}$ should be a c-context, so we are done.

Now, to prove the converse inclusion, we have to prove that given some $s \in [\![e]\!]^{\alpha pl}$ such that $\vdash_{\pi^\alpha CRWL} c\mathcal{C}[s] \twoheadrightarrow t$ then $\vdash_{\pi^\alpha CRWL} c\mathcal{C}[e] \twoheadrightarrow t$. If $c\mathcal{C} = []$ then we have $s \equiv c\mathcal{C}[s] \twoheadrightarrow t$, and so $t \sqsubseteq s$ by Lemma 9 (page 36). But then, as $s \in [\![e]\!]^{\alpha pl}$, we get $c\mathcal{C}[e] \equiv e \twoheadrightarrow t$ by Proposition 1 (page 8). To prove the case where $c\mathcal{C} \neq []$ we proceed by induction on the size of the proof for $\vdash_{\pi^\alpha CRWL} c\mathcal{C}[s] \twoheadrightarrow t$. Concerning the base cases:

B Then $t \equiv \bot$ and we can take $s \equiv \bot$ for which $e \twoheadrightarrow \bot$ and $c\mathcal{C}[s] \twoheadrightarrow \bot \equiv t$ by rule B in both cases.

RR, DC Then either $c\mathcal{C}[e] \equiv X \in \mathcal{V}$ or $c\mathcal{C}[e] \equiv c \in CS^0$, therefore $c\mathcal{C} = []$ and we proceed like in the previous case.

Regarding the inductive step:

DC If $c\mathcal{C} = []$ we proceed like in the previous case, otherwise we have:

$$\frac{e_1 \twoheadrightarrow t_1 \quad \ldots \quad c\mathcal{C}'[s] \twoheadrightarrow t' \quad \ldots \quad e_n \twoheadrightarrow t_n}{c\mathcal{C}[s] \equiv c(e_1, \ldots, c\mathcal{C}'[s], \ldots, e_n) \twoheadrightarrow c(t_1, \ldots, t', \ldots, t_n) \equiv t}\ \mathbf{DC}$$

But then we can apply the IH to $c\mathcal{C}'[s] \twoheadrightarrow t'$ with $s \in [\![e]\!]^{\alpha pl}$ to get $c\mathcal{C}'[e] \twoheadrightarrow t'$, so we can build the following proof:

$$\frac{\dfrac{}{e_1 \twoheadrightarrow t_1}\,hypothesis \quad \ldots \quad \dfrac{}{c\mathcal{C}'[e] \twoheadrightarrow t'}\,IH \quad \ldots \quad \dfrac{}{e_n \twoheadrightarrow t_n}\,hypothesis}{c\mathcal{C}[e] \equiv c(e_1, \ldots, c\mathcal{C}'[e], \ldots, e_n) \twoheadrightarrow c(t_1, \ldots, t', \ldots, t_n) \equiv t}\ \mathbf{DC}$$

POR$^\alpha$ If $c\mathcal{C} = []$ we proceed like in the previous case, otherwise we have $c\mathcal{C} = f(e_1, \ldots, c\mathcal{C}', \ldots, e_n)$ for some $f \in FS$, which contradicts the hypothesis as $c\mathcal{C}$ should be a c-context, so we are done.

□

Lemma 11. *Under any program and for every $e_1, e_2 \in Exp_\perp$, $[\![e_1 \ ?\ e_2]\!]^{\alpha pl} = [\![e_1]\!]^{\alpha pl} \cup [\![e_2]\!]^{\alpha pl}$.*

Proof. For the left to right inclusion, assume we have $\vdash_{\pi^\alpha CRWL} e_1 \ ?\ e_2 \twoheadrightarrow t$, then it must be with a proof of the following shape

$$\frac{\begin{array}{llll} e_1 \twoheadrightarrow s_{11} & e_2 \twoheadrightarrow s_{21} & & \\ \ldots & \ldots & & \\ e_1 \twoheadrightarrow s_{1m_1} & e_2 \twoheadrightarrow s_{2m_2} & s_{i1}\ ?\ (s_{i2}\ ?\ \ldots\ ?\ s_{im_i}) \twoheadrightarrow t & \end{array}}{e_1 \ ?\ e_2 \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

As any $\pi^\alpha CRWL$-proof is finite, then either $m_i = 1$ or the proof for $\vdash_{\pi^\alpha CRWL} s_{i1}\ ?\ (s_{i2}\ ?\ \ldots\ ?\ s_{im_i}) \twoheadrightarrow t$ ends with a proof for $\vdash_{\pi^\alpha CRWL} s \twoheadrightarrow t$ as premise, for some $s \in [\![s_{ij}]\!]^{\alpha pl}$ for some $j \in \{1, \ldots, m_i\}$. Anyway by Lemma 9 we get $t \sqsubseteq s \sqsubseteq s_{ij} \in [\![e_i]\!]^{\alpha pl}$ (with $s \equiv s_{ij}$ in case $m_i = 1$). But then $t \in [\![e_i]\!]^{\alpha pl} \subseteq [\![e_1]\!]^{\alpha pl} \cup [\![e_2]\!]^{\alpha pl}$ by Proposition 1.
Regarding the converse inclusion, assume $t \in [\![e_i]\!]^{\alpha pl}$, then we can build the following proof:

$$\frac{\dfrac{}{e_i \twoheadrightarrow t}\,hypothesis \quad \dfrac{}{e_{3-i} \twoheadrightarrow \perp}\ \mathbf{B} \quad \dfrac{}{t \twoheadrightarrow t}\,\text{Lemma 9}}{e_1 \ ?\ e_2 \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

□

For Lemma 2. Consider some element $(t_1, \ldots, t_n) \in \{(X_1\theta, \ldots, X_n\theta) \mid \theta \in \Theta\}$ and its corresponding $\theta \in \Theta$. We can just take $\theta_1 = \ldots = \theta_n = \theta$ to conclude that $(t_1, \ldots, t_n) \equiv (X_1\theta, \ldots, X_n\theta) \equiv (X_1\theta_1, \ldots, X_n\theta_n) \in \{X_1\theta_1\} \times \ldots \times \{X_n\theta_n\} \subseteq \{X_1\theta_1 \mid \theta_1 \in \Theta\} \times \ldots \times \{X_n\theta_n \mid \theta_n \in \Theta\}$. □

A.2 For Section 4.1

Proof for Lemma 4. First of all Θ is required to be finite and not empty because otherwise $?\Theta$ is not defined. Given $X \in \mathcal{V}$ assume $\vdash_{\pi^\beta CRWL} (?\Theta)(X) \twoheadrightarrow t$ for some $t \in CTerm_\perp$, then we have two possibilities:

a) $\exists \theta_i \in \Theta. X \notin dom(\theta_i)$. Then the hypothesis is $\vdash_{\pi^\beta CRWL} (?\Theta)(X) \equiv X\ ?\ \rho_1(X)\ ?\ \ldots\ ?\ \rho_m(X) \twoheadrightarrow t$ for some $\{\rho_1, \ldots, \rho_m\} \subseteq \Theta$, and we have two possibilities:

   i) If $\vdash_{\pi^\beta CRWL} X \twoheadrightarrow t$ then either $t \equiv \perp$, so $\vdash_{\pi^\alpha CRWL} \sigma(X) \twoheadrightarrow \perp \equiv t$ by **B**; or $t \equiv X$. But then $\vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow \theta_i(X) \equiv$, because $X \notin dom(\theta_i)$ and $\theta_i \in \Theta \subseteq [\![\sigma]\!]^{\beta pl}$.

   ii) On the other hand $\vdash_{\pi^\beta CRWL} \rho_j(X) \twoheadrightarrow t$ for some $\theta_j \in \{\rho_1, \ldots, \rho_m\}$, hence $t \sqsubseteq \rho_j(X)$ by Lemma 9. Besides $\vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow \rho_j(X)$, because $\theta_j \in \{\rho_1, \ldots, \rho_m\} \subseteq \Theta \subseteq [\![\sigma]\!]^{\beta pl}$. But then $\vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow t$ by the polarity of $\pi^\beta CRWL$ from Theorem 2.

b) Otherwise $\forall \theta_i \in \Theta. X \in dom(\theta_i)$. Then if $\Theta = \{\theta_1, \ldots, \theta_n\}$ the hypothesis is $\vdash_{\pi^\beta CRWL} (?\Theta)(X) \equiv \theta_1(X)\ ?\ \ldots\ ?\ \theta_n(X) \twoheadrightarrow t$ and so $\vdash_{\pi^\alpha CRWL} \theta_i(X) \twoheadrightarrow t$ for some $\theta_i \in \Theta$, and we can proceed like in the second item of the previous case.

□

Lemma 12. *For any program $\mathcal{P}$, $\sigma \in Subst_\perp$ we have that $[\![\sigma]\!]^{\beta pl} \neq \emptyset$ and given $\overline{X} = dom(\sigma)$ then $[\overline{X/\perp}] \in [\![\sigma]\!]^{\beta pl}$.*

Proof. It is enough to prove that if $\overline{X} = dom(\sigma)$ then $[\overline{X/\ \bot}] \in [\![\sigma]\!]^{\alpha pl}$. First of all $[\overline{X/\ \bot}] \in CSubst_\bot$ by definition. Now consider some $Y \in \mathcal{V}$.

i) If $Y \in \overline{X}$ then $\vdash_{\pi^\alpha CRWL} \sigma(Y) \twoheadrightarrow \bot \equiv Y[\overline{X/\ \bot}]$, by rule **B**.

ii) Otherwise $Y \notin \overline{X} = dom(\sigma)$, hence $\vdash_{\pi^\alpha CRWL} \sigma(Y) \equiv Y \twoheadrightarrow Y \equiv Y[\overline{X/\ \bot}]$, by rule **RR**.

□

Lemma 13. *For any finite and not empty $\Theta, \Theta' \subseteq CSubst_\bot$, if $\Theta \subseteq \Theta'$ then $?\Theta \sqsubseteq_\pi ?\Theta'$.*

Proof. Let $\Theta = \{\theta 1, \ldots, \theta_n\}$ and $\Theta' = \{\theta'_1, \ldots, \theta'_k\}$ be. Given an arbitrary $X \in \mathcal{V}$:

1. If $\exists \theta \in \Theta . X \notin dom(\theta)$ then $\theta \in \Theta'$, because $\Theta \subseteq \Theta'$, which also implies that for $\rho_1, \ldots, \rho_m = \theta_1, \ldots, \theta_n \mid \lambda\theta.X \in dom(\theta)$ and $\rho'_1, \ldots, \rho'_l = \theta'_1, \ldots, \theta'_k \mid \lambda\theta.X \in dom(\theta)$ then $\{\rho_1, \ldots, \rho_m\} \subseteq \{\rho'_1, \ldots, \rho'_l\}$. But then
$$(?\Theta)(X) = X\ ?\ \rho_1(X)\ ?\ \ldots\ ?\ \rho_m(X)$$
$$(?\Theta')(X) = X\ ?\ \rho'_1(X)\ ?\ \ldots\ ?\ \rho'_l(X)$$
and $\{X, \rho_1(X), \ldots, \rho_m(X)\} \subseteq \{X, \rho'_1(X), \ldots, \rho'_l(X)\}$, so $?\Theta \sqsubseteq_\pi ?\Theta'$ holds for X by reflexivity of $\sqsubseteq$.

2. Otherwise $\forall \theta \in \Theta . X \in dom(\theta)$ which implies $(?\Theta)(X) = \theta_1(X)\ ?\ \ldots\ ?\ \theta_n(X)$. But then for $\rho'_1, \ldots, \rho'_l = \theta'_1, \ldots, \theta'_k \mid \lambda\theta.X \in dom(\theta)$ we have that $\Theta \subseteq \Theta'$ implies both $\{\theta_1(X), \ldots, \theta_n(X)\} \subseteq \{X, \rho'_1(X), \ldots, \rho'_l(X)\}$ and $\{\theta_1(X), \ldots, \theta_n(X)\} \subseteq \{\theta'_1(X), \ldots, \theta'_k(X)\}$, so $?\Theta \sqsubseteq_\pi ?\Theta'$ holds for X by reflexivity of $\sqsubseteq$.

□

Proof for Lemma 5. By a case distinction over e:

- If $e \equiv X \in dom(\sigma)$: Then $e\sigma \equiv \sigma(X) \twoheadrightarrow t$, so we can define:
$$\theta(Y) = \begin{cases} t & \text{if } Y \equiv X \\ \bot & \text{if } Y \in (dom(\sigma) \setminus \{X\}) \\ Y & \text{if } Y \notin dom(\sigma) \end{cases}$$
Then $\theta \in [\![\sigma]\!]^{\alpha pl}$ because obviously $\theta \in CSubst_\bot$, and given $Z \in \mathcal{V}$.
  a) If $Z \equiv X$ then $\vdash_{\pi^\alpha CRWL} \sigma(Z) \equiv \sigma(X) \twoheadrightarrow t \equiv \theta(Z)$ by hypothesis.
  b) If $Z \in (dom(\sigma) \setminus \{X\})$ then $\vdash_{\pi^\alpha CRWL} \sigma(Z) \twoheadrightarrow \bot \equiv \theta(Z)$ by rule **B**.
  c) Otherwise $Z \notin dom(\sigma)$ and then $\vdash_{\pi^\alpha CRWL} \sigma(Z) \equiv Z \twoheadrightarrow Z \equiv \theta(Z)$ by rule **RR**.

  Now we can take $\Theta = \{\theta\}$ for which $e(?\Theta) \equiv \theta(X) \equiv t \twoheadrightarrow t$, by Lemma 9.

- If $e \equiv X \notin dom(\sigma)$: Then given $\overline{Y} = dom(\sigma)$ we define $[\overline{Y/\ \bot}]$ for which $[\overline{Y/\ \bot}] \in [\![\sigma]\!]^{\alpha pl}$ by Lemma 12, so we can take $\Theta = \{[\overline{Y/\ \bot}]\}$ for which $[\![e\sigma]\!]^{\alpha pl} = [\![X]\!]^{\alpha pl} = [\![X(?\Theta)]\!]^{\alpha pl}$.

- If $e \notin \mathcal{V}$ then we proceed by induction over the structure of $e\sigma \twoheadrightarrow t$:

  Base cases

  B Then $t \equiv \bot$, so given $\overline{Y} = dom(\sigma)$ we can take $\Theta = \{[\overline{Y/\ \bot}]\}$ for which $e(?\Theta) \twoheadrightarrow \bot$ by **B**.

  RR Then $e \in \mathcal{V}$ and we are in the previous case.

  DC Similar to the case for $e \equiv X \notin dom(\sigma)$.

  Inductive steps

  DC Then $e \equiv c(e_1, \ldots, e_n)$, as $e \notin \mathcal{V}$, and we have:
$$\frac{e_1\sigma \twoheadrightarrow t_1 \quad \ldots \quad e_n\sigma \twoheadrightarrow t_n}{e\sigma \equiv c(e_1\sigma, \ldots, e_n\sigma) \twoheadrightarrow c(t_1, \ldots, t_n) \equiv t}\ \mathbf{DC}$$
  By IH of the proof of the other cases $\forall i \in \{1, \ldots, n\}\ \exists \Theta_i \subseteq [\![\sigma]\!]^{\alpha pl}$ such that $e_i(?\Theta_i) \twoheadrightarrow t_i$. Hence we can define $\Theta = \bigcup_{i \in \{1,\ldots,n\}} \Theta_i$, for which $\forall i \in \{1, \ldots, n\}\ ?\Theta_i \sqsubseteq_\pi ?\Theta$ by Lemma 13, and so $\forall i \in \{1, \ldots, n\}\ e_i(?\Theta_i) \twoheadrightarrow t_i$ implies $e_i(?\Theta) \twoheadrightarrow t_i$, by the monotonicity for substitutions of $\pi^\beta CRWL$ from Theorem 2. Therefore $e(?\Theta) \twoheadrightarrow c(t_1, \ldots, t_n)$ by **DC**.

POR$^\beta$ Similar to the proof of the previuos case. We also have $e \equiv f(e_1, \ldots, e_n)$ (as $e \notin \mathcal{V}$) and given a proof for $e\sigma \equiv f(e_1, \ldots, e_n)\sigma \twoheadrightarrow t$, we apply the IH of the proof of the other cases to every $e_i\sigma \twoheadrightarrow p_i\theta_{ij}$ to get some $\Theta_{ij} \subseteq [\![\sigma]\!]^{\alpha pl}$ such that $e_i(?\Theta_{ij}) \twoheadrightarrow p_i\theta_{ij}$. Then we define $\Theta = \bigcup_{i\in\{1,\ldots,n\}} \bigcup_{j\in\{1,\ldots,m_i\}} \Theta_{ij}$ for which $\forall i, j,\ ?\Theta_{ij} \sqsubseteq_\pi ?\Theta$ by Lemma 13, and as a consequence $\forall i, j,\ e_i(?\Theta) \twoheadrightarrow p_i\theta_{ij}$. Hence with $e(?\Theta) \equiv f(e_1(?\Theta), \ldots, e_n(?\Theta))$ we can compute the same value for its arguments and thus use the same compressible substitution $\theta \in \mathit{CSubst}^?_\perp$ for parameter passing in **POR**$^\beta$, employing the same $\theta_e \in \mathit{CSubst}^?_\perp$ to instantiate the extra variables.

□

Proof for Proposition 5. a) By definition, as $\Theta \subseteq \mathit{CSubst}_\perp$ and is also finite.

b) Assuming $\Theta \subseteq cc(\Theta)$ then $?\Theta \sqsubseteq_\pi ?cc(\Theta)$ holds by Lemma 13. So all that is left is proving that $\Theta \subseteq cc(\Theta)$. Conside some $\theta \in \Theta$, then $dom(\theta) \subseteq \{X_1, \ldots, X_n\}$ by definition of $\{X_1, \ldots, X_n\}$, hence $\theta = [X_1\theta, \ldots, X_n\theta] \in cc(\Theta)$, by definition of $cc(\Theta)$, as $\theta \in \Theta$.

c) For any $\mu \in cc(\Theta)$ we have that $\mu = [X_1/X_1\theta_1, \ldots, X_n/X_n\theta_n]$ for some $\theta_1, \ldots, \theta_n \in \Theta$, by definition of $cc(\Theta)$. Therefore $dom(\mu) \subseteq \{X_1, \ldots, X_n\}$, thus $\bigcup_{\mu\in cc(\Theta)} dom(\mu) \subseteq \{X_1, \ldots, X_n\}$.
Regarding the converse inclusion, given some $X_1 \in \{X_1, \ldots, X_n\}$ by definition of $\{X_1, \ldots, X_n\}$ then $\exists\theta \in \Theta$ such that $X_i \in dom(\theta)$, thus $\theta(X_i) \not\equiv X_i$. Moreover $[X_1/X_1\theta, \ldots, X_i/X_i\theta, \ldots, X_n/X_n\theta] \in cc(\Theta)$, by definition of $cc(\Theta)$, and $[X_1/X_1\theta,\ \ldots, X_i/X_i\theta, \ldots, X_n/X_n\theta](X_i) \equiv \theta(X_i) \not\equiv X_i$ by the previous fact, therefore $X_i \in dom([X_1/X_1\theta, \ldots, X_i/X_i\theta, \ldots, X_n/X_n\theta])$, which implies $X_i \in \bigcup_{\mu\in cc(\Theta)} dom(\mu)$, thus $\{X_1, \ldots, X_n\} \subseteq \bigcup_{\mu\in cc(\Theta)} dom(\mu)$

d) By the previous item we have that $\bigcup_{\mu\in cc(\Theta)} dom(\mu) = \{X_1, \ldots, X_n\}$. We will use the criterion from Lemma 2, so let us take some $\mu_1, \ldots, \mu_n \in cc(\Theta)$. By definition of $cc(\Theta)$ we have that for each μ_i $\exists\theta_i \in \Theta$ such that $X_i\mu_i \equiv X_i\theta_i$. But then $(X_1\mu_1, \ldots, X_n\mu_n) \equiv (X_1\theta_1, \ldots, X_n\theta_n) \equiv (X_1, \ldots, X_n)[X_1/X_1\theta_1, \ldots, X_n/X_n\theta_n]$, and $[X_1/X_1\theta_1, \ldots, X_n/X_n\theta_n] \in cc(\Theta)$ by definition of $cc(\Theta)$, so we are done.

□

Proof for Lemma 6. Let $\bigcup_{\theta\in\Theta} dom(\theta) = \{X_1, \ldots, X_n\} = \bigcup_{\mu\in cc(\Theta)} dom(\mu)$ by Proposition 5, and let us consider some $\mu \in cc(\Theta)$. Then $\mu = [X_1/X_1\theta_1, \ldots, X_n/X_n\theta_n]$ for some $\theta_1, \ldots, \theta_n \in \Theta$. Then given some $X \in \mathcal{V}$ we have the following possibilities:

a) $X \equiv X_i \in \{X_1, \ldots, X_n\}$: Then $\vdash_{\pi^\beta CRWL} \sigma(X) \equiv \sigma(X_i) \twoheadrightarrow \theta_i(X_i)$, as $\theta_i \in \Theta \subseteq [\![\sigma]\!]^{\beta pl}$. But $X \equiv X_i \in \{X_1, \ldots, X_n\}$ implies $\theta_i(X_i) \equiv \mu(X_i) \equiv \mu(X)$, so $\vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow \mu(X)$.

b) $X \notin \{X_1, \ldots, X_n\}$: Then we can take any $\theta_i \in \Theta$ (it must exist since Θ is not empty) so $\vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow \theta_i(X_i)$, as $\theta_i \in \Theta \subseteq [\![\sigma]\!]^{\beta pl}$. But then $X \notin dom(\theta_i)$, as $\{X_1, \ldots, X_n\} = \bigcup_{\theta\in\Theta} dom(\theta) \supseteq dom(\theta_i)$, therefore $\theta_i(X) \equiv X \equiv \mu(X)$, as $dom(\mu) \subseteq \bigcup_{\mu\in cc(\Theta)} dom(\mu) = \{X_1, \ldots, X_n\}$, hence $\vdash_{\pi^\beta CRWL} \sigma(X) \twoheadrightarrow \mu(X)$.

Thus we have proved that $\forall U \in \mathcal{V}, \vdash_{\pi^\beta CRWL} \sigma(U) \twoheadrightarrow \mu(U)$, henceforth $\mu \in [\![\sigma]\!]^{\beta pl}$ and so $cc(\Theta) \subseteq [\![\sigma]\!]^{\beta pl}$. □

Lemma 14. *For any $t \in \mathit{CTerm}_\perp, \Theta \subseteq \mathit{CSubst}_\perp$ finite, given $\theta_i \in \Theta$ then $\vdash_{\pi^\alpha CRWL} t(?\Theta) \twoheadrightarrow t\theta_i$.*

Proof. A simple induction on the structure of t. □

Lemma 15. *For any $\Theta \subseteq \mathit{CSubst}_\perp$ compressible and $\mathcal{D} \subseteq \mathcal{V}$ we have that $\Theta' = \{\theta|_\mathcal{D} \mid \theta \in \Theta\}$ is also compressible.*

Proof. Consider $\{Y_1, \ldots, Y_m\} = \bigcup_{\theta'\in\Theta'} dom(\theta')$, by Lemma 2 all we have to prove is that $\{(Y_1\theta', \ldots, Y_m\theta') \mid \theta' \in \Theta'\} \supseteq \{Y_1\theta' \mid \theta' \in \Theta'\} \times \ldots \times \{Y_m\theta' \mid \theta' \in \Theta'\}$. To consider some element of the latter set let us fix some $\theta'_1, \ldots, \theta'_m \in \Theta'$ so we have $(Y_1\theta'_1, \ldots, Y_m\theta'_m)$. Now for $\overline{X} = \bigcup_{\theta\in\Theta} dom(\theta)$, then by definition of Θ' we get that $\forall j \in \{1, \ldots, m\}.\exists\theta_j \in \Theta, X_j \in \overline{X}$ such that $\theta'_j = \theta_j|_\mathcal{D}$ and $X_j \equiv Y_j$, so $(Y_1\theta'_1, \ldots, Y_m\theta'_m)$ is in

a sense contained in some tuple $(X_1\theta_1, \ldots, X_n\theta_n)$, as it is a subtuple of this later tuple. Then, as Θ is compressible then $\exists\theta \in \Theta$ such that $\forall j \in \{1, \ldots, m\}.X_j\theta_j \equiv X_j\theta$, but then for each $j \in \{1, \ldots, m\}$

$$\begin{array}{ll} Y_j\theta'_j \equiv X_j\theta_j|_{\mathcal{D}} & \text{by definition of } \Theta \\ \equiv X_j\theta_j & \text{as } X_j \equiv Y_j \in \mathcal{D} \text{ by definition of } \{Y_1, \ldots, Y_m\} \\ \equiv X_j\theta & \text{taking } \theta \text{ above} \\ \equiv Y_j\theta & \text{as } X_j \equiv Y_j \\ \equiv Y_j\theta|_{\mathcal{D}} \text{ as } Y_j \in \mathcal{D} & \end{array}$$

But $\theta \in \Theta$ and so $\theta|_{\mathcal{D}} \in \Theta'$ by definition, and we are done as then $(Y_1\theta'_1, \ldots, Y_m\theta'_m) = (Y_1\theta|_{\mathcal{D}}, \ldots, Y_m\theta|_{\mathcal{D}}) \in \{(Y_1\theta', \ldots, Y_m\theta') \mid \theta' \in \Theta'\}$. □

Proof for Lemma 3. Let us assume the term rewriring step was performed at the root of the expression, so it has the shape $e \equiv f(\overline{p})\sigma \to r\sigma \equiv e'$. Thus, given some $t \in [\![r\sigma]\!]^{\beta pl}$ our goal is proving that $t \in [\![f(\overline{p})\sigma]\!]^{\beta pl}$. First of all by Lemma 7 we get some compressible $\Theta \subseteq [\![\sigma]\!]^{\beta pl}$ such that $t \in [\![r(?\Theta)]\!]^{\beta pl}$. If we could use it to prove that $t \in [\![f(\overline{p})(?\Theta)]\!]^{\beta pl}$ then by Lemma 4 we would get $?\Theta \trianglelefteq \sigma$, so by the monotonicity of Theorem 2 we could obtain $t \in [\![f(\overline{p})\sigma]\!]^{\beta pl}$ as we wanted. As $\overline{p} \subseteq CTerm_\perp$ and $\Theta \subseteq CSubst_\perp$ then by Lemma 14 we get that $\forall p_i \in \overline{p}, \theta_j \in \Theta$ we have $\vdash_{\pi^\beta CRWL} p_i(?\Theta) \twoheadrightarrow p_i\theta_j$. All this can be used to perform the following step, assuming $\Theta = \{\theta_1, \ldots, \theta_m\}$.

$$\frac{\begin{array}{ccccc} p_1(?\Theta) \twoheadrightarrow p_1\theta_1 \equiv p_1\theta_1|_{var(p_1)} & & & & p_n(?\Theta) \twoheadrightarrow p_n\theta_1 \equiv p_n\theta_1|_{var(p_n)} \\ \cdots & & \cdots & & \cdots \\ p_1(?\Theta) \twoheadrightarrow p_1\theta_m \equiv p_1\theta_m|_{var(p_1)} & & & & p_n(?\Theta) \twoheadrightarrow p_n\theta_m \equiv p_n\theta_m|_{var(p_n)} \quad r\theta' \equiv r(?\Theta) \twoheadrightarrow t \end{array}}{f(p_1, \ldots, p_n)(?\Theta) \twoheadrightarrow t}\ \mathbf{POR}^\beta$$

for $\theta' = (\biguplus ?\Theta_i) \uplus \theta_e$ where $\forall i \in \{1, \ldots, n\}.\Theta_i = \{\theta_j|_{var(p_i)} \mid \theta_j \in \Theta\}$, $\theta_e = (?\Theta)|_{\mathcal{V}_e}$ for $\mathcal{V}_e = vExtra(f(\overline{p}) \to r)$. Note that $\forall i.dom(?\Theta_i) \subseteq var(p_i)$ by Lemma 1, therefore θ' is well defined by left linearity of program rules. As Θ is compressible then each Θ_i is also compressible, by Lemma 15, and so they can be used to build θ' in the $\mathbf{POR}^\beta$ step above. All that is left is proving $r\theta' \equiv r(?\Theta)$, which holds because given any $X \in var(r)$ we have $\theta'(X) \equiv (?\Theta)(X)$. As $var(r) \subseteq \biguplus_{i=1}^{n} var(p_i) \uplus \mathcal{V}_e$ then we have two possibilities for such $X \in var(r)$:

a) $\exists! p_i \in \overline{p}.X \in var(p_i)$ then $\theta'(X) \equiv (?\Theta_i)(X)$. To prove that $(?\Theta_i)(X) \equiv (?\Theta)(X)$ note that as $X \in var(p_i)$ then $\forall \theta_j \in \Theta$ we have that $X \in dom(\theta_j)$ implies $X \in dom(\theta_j)|_{var(p_i)}$, therefore $\forall \theta_j \in \Theta.X \in dom(\theta_j)$ iff $X \in dom(\theta_j)|_{var(p_i)}$. Now we have two possibilities:

   i) If $\exists \theta_j \in \Theta.X \notin dom(\theta_j)$ let $\rho_1 \ldots \rho_l = \theta_1 \ldots \theta_m \mid \lambda\theta.(X \in dom(\theta))$. Then $\theta_1|_{var(p_i)} \ldots \theta_m|_{var(p_i)} \mid \lambda\theta.(X \in dom(\theta)) = \rho_1|_{var(p_i)} \cdots \rho_l|_{var(p_i)}$ and so

$$\begin{array}{ll} (?\Theta)(X) \equiv X \ ? \ \rho_1(X) \ ? \ \ldots \ ? \ \rho_l(X) & \\ \equiv X \ ? \ \rho_1|_{var(p_i)}(X) \ ? \ \ldots \ ? \ \rho_l|_{var(p_i)}(X) & \text{as } X \in var(p_i) \\ \equiv (?\Theta_i)(X) & \end{array}$$

   ii) Otherwise $\forall \theta_j \in \Theta.X \in dom(\theta_j)$ and so $\forall \theta_j \in \Theta.X \in dom(\theta_j|_{var(p_i)})$, as $X \in var(p_i)$. But then

$$\begin{array}{ll} (?\Theta)(X) \equiv \theta_1(X) \ ? \ \ldots \ ? \ \theta_m(X) & \\ \equiv \theta_1|_{var(p_i)}(X) \ ? \ \ldots \ ? \ \theta_m|_{var(p_i)}(X) & \text{as } X \in var(p_i) \\ \equiv (?\Theta_i)(X) & \end{array}$$

   Therefore $\theta'(X) \equiv (?\Theta_i)(X) \equiv (?\Theta)(X)$.

b) Otherwise $X \in \mathcal{V}_e$, but then

$$\begin{array}{ll} \theta'(X) \equiv \theta_e(X) & \text{by definition of } \theta' \\ \equiv (?\Theta)|_{\mathcal{V}_e}(X) & \text{by definition of } \theta_e \\ \equiv (?\Theta)(X) & \text{as } X \in \mathcal{V}_e \end{array}$$

We have just proved the soundness w.r.t. $\pi^\beta CRWL$ of term rewriting steps performed at the root of the starting expression. So all that is left is using the compositionality of $\pi^\beta CRWL$ from Theorem 2 for propagating this result for steps performed in an arbitrary context. If the step was not performed at the

root of the expression then we have $e \equiv \mathcal{C}[f(\overline{p})\sigma] \to \mathcal{C}[r\sigma] \equiv e'$ for which $f(\overline{p})\sigma \to r\sigma$ is performed at the top. But then $[\![r\sigma]\!]^{\beta pl} \subseteq [\![f(\overline{p})\sigma]\!]^{\beta pl}$ by the proof of the previous case, and we can chain:

$$\begin{array}{lll} [\![\mathcal{C}[r\sigma]]\!]^{\beta pl} = \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![r\sigma]\!]^{\beta pl}} [\![\mathcal{C}[t_1\ ?\ \ldots\ ?\ t_n]]\!]^{\beta pl} & & \text{by Theorem 2} \\ \subseteq \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![f(\overline{p})\sigma]\!]^{\beta pl}} [\![\mathcal{C}[t_1\ ?\ \ldots\ ?\ t_n]]\!]^{\beta pl} = [\![\mathcal{C}[f(\overline{p})\sigma]]\!]^{\beta pl} & & \text{by Theorem 2} \end{array}$$

□

A.3 For Section 4.2

In the proof for Theorem 6 we will use the following notion of substitution modification to formalize the reduction into $\bot$ of the values that a starting matching substitution $?\Theta_i$, used in a $\mathbf{POR}^\alpha$ step, assigns to the variables in a left-hand side argument p_i which are also different to the at most unique—when not absent—variable $X \in var(p_i) \cap var(r)$, granted because the program belongs to the class $\mathcal{C}^{\alpha\beta}$.

Definition 9 (Substitution modification)**.** *For any $\sigma \in Subst_\bot$ its modification $\sigma\{X/e\} \in Subst_\bot$, in which the value for $X \in \mathcal{V}$ is replaced by some $e \in Exp_\bot$ is defined as*

$$(\sigma\{X/e\})(Y) = \begin{cases} e & \text{if } X \equiv Y \\ \sigma(Y) & \text{otherwise} \end{cases}$$

Lemma 16. *For any $\sigma \in Subst_\bot$, $X \in \mathcal{V}$, $e \in Exp_\bot$ we have*

$$dom(\sigma\{X/e\}) \subseteq dom(\sigma) \cup \{X\}$$

Proof. Given some $Y \in dom(\sigma\{X/e\})$, if $Y \equiv X$ then $Y \in dom(\sigma) \cup \{X\}$. Otherwise $Y \not\equiv X$, which implies $Y \in dom(\sigma)$ as

$$\begin{array}{ll} Y \not\equiv (\sigma\{X/e\})(Y) & \text{as } Y \in dom(\sigma\{X/e\}) \\ \equiv \sigma(Y) & \text{as } Y \not\equiv X \end{array}$$

But then $Y \in dom(\sigma)$ by definition. □

Proof for Theorem 6. By Theorem 3 we just have to prove that $[\![e]\!]^{\alpha pl}_{\mathcal{P}} \subseteq [\![e]\!]^{\beta pl}_{\mathcal{P}}$ whenever $\mathcal{P} \in \mathcal{C}^{\alpha\beta}$. Assume a $\pi^\alpha CRWL$-proof for $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ for some $e \in Exp_\bot$, $t \in CTerm_\bot$, we will show that we can build another $\pi^\alpha CRWL$-proof for the same statement which only uses compressible substitutions in $\mathbf{POR}^\alpha$ steps. As a consequence each $\mathbf{POR}^\alpha$ step will be also a valid $\mathbf{POR}^\beta$ step, so that will be also a $\pi^\beta CRWL$-proof for $\mathcal{P} \vdash_{\pi^\beta CRWL} e \twoheadrightarrow t$.

Therefore all that is left is building the new $\pi^\alpha CRWL$-proof. To do that we proceed by induction on the size of the original $\pi^\alpha CRWL$-proof for $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$, performing a case distinction over the $\pi^\alpha CRWL$ rule applied at the root of the starting proof. The cases for **B** and **RR** are trivial, and so it is the case for **DC**, where we just have to apply the IH over each premise $\mathcal{P} \vdash_{\pi^\alpha CRWL} e_i \twoheadrightarrow t_i$, then building the target $\pi^\alpha CRWL$-proof with an application of **DC** at its root and the resulting proofs from the applications of the IH as its premises.

The interesting case is that for an application of $\mathbf{POR}^\alpha$. For the sake of simplicity we will prove if for $f \in FS^1$, but this proof can be easily extended to functions of arbitrary arity thanks to left linearity of program rules. Then we have the following proof

$$\frac{\begin{array}{l} e \to p\theta_1 \\ \ldots \\ e \to p\theta_m \quad r\theta \twoheadrightarrow t \end{array}}{\mathcal{P} \vdash_{\pi^\alpha CRWL} f(e) \to t}\ \mathbf{POR}^\alpha$$

For some $(f(p) \to r) \in \mathcal{P}$, $\Theta = \{\theta_1, \ldots, \theta_m\}.\forall j.dom(\theta_j) \subseteq var(p)$, $\theta = ?\Theta \uplus \theta_e$ for some $\theta_e \in CSubst^?_\bot$. But as $\mathcal{P} \in \mathcal{C}^{\alpha\beta}$ then we have $\#(var(p) \cap var(r)) = K \leq 1$. Hence either $K = 0$ or $K = 1$, let us consider both cases.

<u>If $K = 0$</u>: We define $\overline{X} \subseteq \mathcal{V}$ as $\overline{X} = \bigcup_{j \in \{1,\ldots,m\}} dom(\theta_j)$. Then we have $\forall j.[\overline{X/\ \bot}] \sqsubseteq \theta_j$ because given an arbitrary $Y \in \mathcal{V}$:

- If $Y \in \overline{X}$ then $[\overline{X/\ \bot}](Y) \equiv \bot \sqsubseteq \theta_j(Y)$.

- Otherwise $Y \notin \overline{X}$. But $dom(\theta_j) \subseteq \overline{X}$ by definition of $\overline{X}$, and so $Y \notin dom(\theta_j)$, hence $[\overline{X/\ \bot}](Y) \equiv Y \sqsubseteq Y \equiv \theta_j(Y)$.

Then we can take an arbitrary $\theta_j \in \{\theta_1, \ldots, \theta_m\}$ for which $[\overline{X/\ \bot}] \sqsubseteq \theta_j$ implies $p[\overline{X/\ \bot}] \sqsubseteq p\theta_j$ (easy to check by induction on the structure of expressions). But then applying the polarity of $\pi^\alpha CRWL$ from Proposition 1 over the statement $e \twoheadrightarrow p\theta_j$ we get $e \twoheadrightarrow p[\overline{X/\ \bot}]$ with a proof of the same size or smaller. Hence we can apply the IH over $e \twoheadrightarrow p[\overline{X/\ \bot}]$ and $r\theta \twoheadrightarrow t$ to get a $\pi^\alpha CRWL$-proof for both statements in which only compressible substitutions are used.

All that is left is proving that by reducing e to $p[\overline{X/\ \bot}]$ we can build a substitution θ' with the same effect over r as θ, using a compressible substitutions in the $\mathbf{POR}^\alpha$ step also. First of all, as $K = 0$ then $var(p) \cap var(r) = \emptyset$. Besides, as $\forall j.dom(\theta_j) \subseteq var(p)$ by hypothesis, then $\overline{X} \subseteq var(p)$ by definition, and so $\overline{X} \cap var(r) = \emptyset$. Furthermore, by Lemma 1 we have $dom(?\Theta) = \bigcup_{j\in\{1,\ldots,m\}} dom(\theta_j) = \overline{X}$, hence $dom(?\Theta) \cap var(r) = \emptyset$. But then if we define $\theta' = ?\{[\overline{X/\ \bot}]\} \uplus \theta_e = [\overline{X/\ \bot}] \uplus \theta_e$ then we can chain

$$\begin{array}{ll} r\theta' \equiv r([\overline{X/\ \bot}] \uplus \theta_e) & \\ \equiv r\theta_e & \text{as } \overline{X} \cap var(r) = \emptyset \\ \equiv r(?\Theta \uplus \theta_e) & \text{as } dom(?\Theta) \cap var(r) = \emptyset \\ \equiv r\theta & \end{array}$$

Besides, the set $\{[\overline{X/\ \bot}]\}$ is compressible because it is singleton—trivial by the characterization of Lemma 2. So we can finally take the $\pi^\alpha CRWL$-proofs obtained by IH to build the following $\pi^\alpha CRWL$-proof that only uses compressible substitutions.

$$\frac{e \twoheadrightarrow p[\overline{X/\ \bot}] \quad r\theta' \equiv r\theta \twoheadrightarrow t}{\mathcal{P} \vdash_{\pi^\alpha CRWL} f(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

<u>If $K = 1$</u>: Let us define $\overline{X} = \bigcup_{j\in\{1,\ldots,m\}} dom(\theta_j)$, for which $\overline{X} \subseteq var(p)$ like in the previous case. As $K = 1$ then $\exists! Y \in \mathcal{V}.var(p) \cap var(r) = \{Y\}$. Then we have $\forall j.[\overline{X/\ \bot}]\{Y/\theta_j(Y)\} \sqsubseteq \theta_j$, because given an arbitrary $Z \in \mathcal{V}$:

- If $Z \equiv Y$ then $([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})(Z) \equiv \theta_j(Y) \sqsubseteq \theta_j(Y) \equiv \theta_j(Z)$.
- If $Z \in \overline{X} \setminus \{Y\}$ then $([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})(Z) \equiv [\overline{X/\ \bot}](Z) \equiv \bot \sqsubseteq \theta_j(Z)$.
- Otherwise $Z \not\equiv Y$ and $Z \notin \overline{X}$, which implies $Z \notin dom(\theta_j)$ by definition of $\overline{X}$, but then $([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})(Z) \equiv [\overline{X/\ \bot}](Z) \equiv Z \sqsubseteq Z \equiv \theta_j(Z)$.

Then $\forall \theta_j \in \{\theta_1, \ldots \theta_m\}$ we have $p[\overline{X/\ \bot}]\{Y/\theta_j(Y)\} \sqsubseteq p\theta_j$ and we can apply the polarity of $\pi^\alpha CRWL$ from Proposition 1 over each statement $e \twoheadrightarrow p\theta_j$ to get $e \twoheadrightarrow p[\overline{X/\ \bot}]\{Y/\theta_j(Y)\}$ with a proof of the same size or smaller. Hence we can apply the IH over each of that proofs and over the proof for $r\theta \twoheadrightarrow t$ to get $\pi^\alpha CRWL$-proofs for each of those statements in which only compressible substitutions are used.

All that is left is proving that by reducing e to those values we can build a substitution θ' with the same effect over r as θ, but also using a compressible substitution in the $\mathbf{POR}^\alpha$ step. This substitution will be

$$\Theta' = \{[\overline{X/\ \bot}]\{Y/\theta_j(Y)\} \mid \theta_j \in \{\theta_1, \ldots, \theta_m\}\}$$

Regarding the first part, as $\overline{X} \subseteq var(p)$ then $dom([\overline{X/\ \bot}]) \subseteq var(p)$. Besides, by Lemma 16 we get $\forall \theta_j.dom([\overline{X/\ \bot}]\{Y/\theta_j(Y)\}) \subseteq dom([\overline{X/\ \bot}]) \cup \{Y\}$, which combined with the previous fact yields $\forall \theta_j.dom([\overline{X/\ \bot}]\ \{\theta_j(Y)\}) \subseteq var(p) \cup \{Y\}$. Then combining this with Lemma 1 we get $dom(?\Theta') = \bigcup_{\theta_j} dom([\overline{X/\ \bot}]\ \{Y/\theta_j(Y)\}) \subseteq var(p) \cup \{Y\}$. But as $var(p) \cap var(r) = \{Y\}$ then $dom(?\Theta') \cap var(r) \subseteq \{Y\}$. Similarly we can prove $dom(?\Theta) \cap var(r) \subseteq \{Y\}$, because $\forall \theta_j.dom(\theta_j) \subseteq var(p)$, which implies $dom(?\Theta) \subseteq var(p)$ by Lemma 1. But then if we define $\theta' = ?\Theta' \uplus \theta_e$ then we can chain

$$\begin{array}{ll} r\theta \equiv r(?\Theta \uplus \theta_e) & \\ \equiv r((?\Theta)|_{\{Y\}} \uplus \theta_e) & \text{as } dom(?\Theta) \cap var(r) \subseteq \{Y\} \\ \equiv r([Y/(?\Theta)(Y)] \uplus \theta_e) & \\ \equiv r([Y/(?\Theta')(Y)] \uplus \theta_e) & (*) \\ \equiv r((?\Theta')|_{\{Y\}} \uplus \theta_e) & \text{as } dom(?\Theta') \cap var(r) \subseteq \{Y\} \\ \equiv r(?\Theta' \uplus \theta_e) \equiv r\theta' & \end{array}$$

$(*)$ as $(?\Theta)(Y) \equiv (?\Theta')(Y)$, because $\forall\theta_j.([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})(Y) \equiv \theta_j(Y)$, therefore $\forall\theta_j.Y \in dom([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})$ iff $Y \in dom(\theta_j)$, which implies that given $\rho_1 \ldots \rho_l = \theta_1 \ldots \theta_m \mid \lambda\theta.(Y \in dom(\theta))$ then $[\overline{X/\ \bot}]\{Y/\theta_1(Y)\} \ldots [\overline{X/\ \bot}]\{Y/\theta_m(Y$ $\mid \lambda\theta.(Y \in dom(\theta)) = [\overline{X/\ \bot}]\{Y/\rho_1(Y)\} \ldots [\overline{X/\ \bot}]\{Y/\rho_l(Y)\}$. But then we have two possibilities:

1. If $\exists\theta_j.Y \notin dom(\theta_j)$ then $Y \notin dom([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})$, therefore
$$\begin{array}{l}(?\Theta)(Y) \equiv Y\ ?\ \rho_1(Y)\ ?\ \ldots\ \rho_l(Y)\\ \equiv Y\ ?\ ([\overline{X/\ \bot}]\{Y/\rho_1(Y)\})(Y)\ ?\ \ldots\ ([\overline{X/\ \bot}]\{Y/\rho_l(Y)\})(Y)\\ \equiv (?\Theta')(Y)\end{array}$$
2. Otherwise $\forall\theta_j.Y \in dom(\theta_j)$, thus $\forall\theta_j.Y \in dom([\overline{X/\ \bot}]\{Y/\theta_j(Y)\})$, therefore
$$\begin{array}{l}(?\Theta)(Y) \equiv \theta_1(Y)\ ?\ \ldots\ \theta_m(Y)\\ \equiv ([\overline{X/\ \bot}]\{Y/\theta_1(Y)\})(Y)\ ?\ \ldots\ ([\overline{X/\ \bot}]\{Y/\theta_m(Y)\})(Y)\\ \equiv (?\Theta')(Y)\end{array}$$

We will now use the characterization of Lemma 2 to prove that Θ' is compressible. Let us define $\overline{U} = \{U_1, \ldots, U_k\} = \bigcup_{\theta' \in \Theta'} dom(\theta')$. By definition of Θ' each θ'_j is of the form $\theta'_j = [\overline{X/\ \bot}]\{Y/\theta_j(Y)\}$. Now we have two possibilities:

1. If $Y \in \overline{U}$ then $Y \equiv U_l$ for some $U_l \in \overline{U}$. By Lemma 2 we can prove that Θ' is compressible if we can find some $\theta' \in \Theta'$ such that $\forall j \in \{1, \ldots, k\}.U_j\theta' \equiv U_j\theta'_j$. We can take $\theta' = \theta'_l$, for which given an arbitrary $U_j \in \overline{U}$:
   (a) If $j = l$ then $U_j\theta' \equiv U_l\theta'_l \equiv U_j\theta'_j$.
   (b) Otherwise $j \neq l$, but then
$$\begin{array}{ll}U_j\theta'_j \equiv U_j([\overline{X/\ \bot}]\{Y/\theta_j(Y)\}) & \\ \equiv U_j[\overline{X/\ \bot}] & \text{as } j \neq l, \text{ which implies } U_j \not\equiv U_l \equiv Y\\ \equiv U_j([\overline{X/\ \bot}]\{Y/\theta_l(Y)\}) & \text{as } U_j \not\equiv Y\\ \equiv U_j\theta'_l \equiv U_j\theta' & \end{array}$$
2. Otherwise $Y \notin \overline{U}$, and then we can take $\theta' = \theta'_j$ for some arbitrary $\theta'_j \in \Theta'$, and proceed like in the second item of the previous case.

Now, we can finally take the $\pi^\alpha CRWL$-proofs obtained by IH to build the following $\pi^\alpha CRWL$-proof that only uses compressible substitutions.

$$\dfrac{\begin{array}{ll}e \twoheadrightarrow [\overline{X/\ \bot}]\{Y/\theta_1(Y)\} & \\ \ldots & \\ e \twoheadrightarrow [\overline{X/\ \bot}]\{Y/\theta_m(Y)\} & r\theta' \equiv r\theta \twoheadrightarrow t\end{array}}{\mathcal{P} \vdash_{\pi^\alpha CRWL} f(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

□

A.4 For Section 4.3

We will use the following simple auxiliary lemma for proving Theorem 7.

Lemma 17. *For any CRWL-program $\mathcal{P}$ and any $\pi^\alpha CRWL$-statement if e_1 then $e_2 \twoheadrightarrow t$ there is a $\pi^\alpha CRWL$-proof for that statement of the form:*

$$\dfrac{e_1 \twoheadrightarrow true \quad e_2 \twoheadrightarrow t \quad t \twoheadrightarrow t}{if\ e_1\ then\ e_2 \twoheadrightarrow t}\ \mathbf{POR}$$

Proof. As the only rule for *if_then_* is *if true then* $X \to X$ then every proof must be of the form:

$$\dfrac{\begin{array}{lll}e_1 \twoheadrightarrow true\theta_1 & e_2 \twoheadrightarrow t_1 & \\ \ldots & \ldots & t_1?\ldots?t_l \twoheadrightarrow t\\ e_1 \twoheadrightarrow true\theta_m & e_2 \twoheadrightarrow t_l & \end{array}}{if\ e_1\ then\ e_2 \twoheadrightarrow t}\ \mathbf{POR}$$

As $t_1\ ?\ \ldots\ ?\ t_l \twoheadrightarrow t$ there must exist some $t_i \in \{t_1, \ldots, t_l\}$ such that $t_i \twoheadrightarrow t$, but then $t \sqsubseteq t_i$ by Lemma 9 and so $e_2 \twoheadrightarrow t_i$ implies $e_2 \twoheadrightarrow t$ by Proposition 1, and we can apply Lemma 9 again to get $t \twoheadrightarrow t$ and construct the proof of the formulation. □

Now we are ready to prove Theorem 7.

Proof for Theorem 7. As seen in Section 4.3, the second part follows from combining the first part with Corollary 4.1, because then we can chain $[\![e]\!]^{rt}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{\alpha pl}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{\alpha pl}_{\mathcal{P}}$.

So all that is left is proving the first part. Let us assume $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ for some $t \in CTerm_\bot$, we will see that then $\mathcal{P} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ by induction on the size of $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$. The base cases are trivial because no program rule is involved, and so it is the case for **DC**, in which we only have to apply the IH over the hypothesis. The case for $\mathbf{POR}^\alpha$ when $e \equiv f(\overline{e})$ and $f \in \{\_?\_, \mathit{if\_then\_}\}$ can be resolved applying the IH too, so the difficult case is that in which $f \notin \{\_?\_, \mathit{if\_then\_}\}$. For the sake of simplicity we will consider $f \in FS^1$, the proof can be easily extended to functions with zero or more than one arguments. Assume the rule used was $f(Y) \to \mathit{if}\ \mathit{match}(Y)\ \mathit{then}\ r[\overline{X_j/\mathit{project}_j(Y)}]$, corresponding to the original rule $f(p) \to r$ and with the auxiliary functions defined by $\mathit{match}(p) \to \mathit{true}$, $\mathit{project}_j(p) \to X_j$, for $\overline{X_j} = var(p) \cap var(r)$. Then the proof has the shape:

$$\frac{\begin{array}{l} e \twoheadrightarrow t_1 \\ \dots \\ e \twoheadrightarrow t_m \quad \mathit{if}\ \mathit{match}(t_1?\dots?t_m)\ \mathit{then}\ r[\overline{X_j/\mathit{project}_j(t_1?\dots?t_m)}] \twoheadrightarrow t \end{array}}{pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} f(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

where

$$\frac{\dfrac{t_1?\dots?t_m \twoheadrightarrow p\mu \quad \mathit{true} \twoheadrightarrow \mathit{true}}{\mathit{match}(t_1?\dots?t_m) \twoheadrightarrow \mathit{true}}\ \mathbf{POR}^\alpha \quad r[\overline{X_j/\mathit{project}_j(t_1?\dots?t_m)}] \twoheadrightarrow t \quad t \twoheadrightarrow t}{pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} \mathit{if}\ \mathit{match}(t_1?\dots?t_m)\ \mathit{then}\ r[\overline{X_j/\mathit{project}_j(t_1?\dots?t_m)}] \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

by Lemma 17. Let $s_1 \dots s_l = t_1 \dots t_m \mid \lambda t.(\exists \theta \in CSubst_\bot . t \equiv p\theta \ \wedge\ dom(\theta) \subseteq var(p))$. These $s_1 \dots s_l$ are the useful values computed for e in the starting $\pi^\alpha CRWL$-proof, that is, those that can be proyected later. We will use those θ_j corresponding to each s_j soon.

Now we will see that $s_1 \dots s_l$ is not empty. As $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} t_1\ ?\ \dots\ ?\ t_m \twoheadrightarrow p\mu$ then $\exists j \in \{1, \dots, m\}$ such that $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} t_j \twoheadrightarrow p\mu$, and so $p\mu \sqsubseteq t_j$ by Lemma 9. But then it is easy to prove by induction on the structure of p, taking advantage of its linearity and totality, that there must exist some $\theta_j \in CSubst_\bot$ such that $t_j \equiv p\theta_j$. Hence $s_1 \dots s_l$ is not empty and then it is easy to prove that $[\overline{X_j/\mathit{project}_j(t_1?\dots?t_m)}]$ and $[\overline{X_j/\mathit{project}_j(s_1?\dots?s_l)}]$ verify the conditions to apply the strong monotonicity of Proposition 2 in order to get that $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} r[\overline{X_j/\mathit{project}_j(s_1?\dots?s_l)}] \twoheadrightarrow t$ with a proof of the same size or smaller. By definition $s_1 \dots s_l \equiv p\theta_1 \dots p\theta_l$, now we will see that the substitutions $[\overline{X_j/\mathit{project}_j(s_1?\dots?s_l)}]$ and $?\{\theta_1, \dots, \theta_l\}|_{var(r)}$ also verify the conditions of to apply the strong monotonicity of Proposition 2. As $\forall \theta \in \{\theta_1, \dots, \theta_l\}.dom(\theta) \subseteq var(p)$ then $dom(?\{\theta_1, \dots, \theta_l\}|_{var(r)}) \subseteq \overline{X_j}$, so given $X \notin \overline{X_j}$ both substitutions leave it untouched. On the other hand if $X \equiv X_j \in \overline{X_j}$, given

$$\frac{\begin{array}{l} s_1\ ?\ \dots\ ?\ s_l \twoheadrightarrow p\mu_1 \\ \dots \\ s_1\ ?\ \dots\ ?\ s_l \twoheadrightarrow p\mu_h \quad X_j(?\{\mu_1, \dots, \mu_h\}) \twoheadrightarrow t \end{array}}{pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} X_j[\overline{X_j/\mathit{project}_j(s_1\ ?\ \dots\ ?\ s_l)}] \equiv \mathit{project}_j(s_1\ ?\ \dots\ ?\ s_l) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

Then $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} X_j(?\{\mu_1, \dots, \mu_h\}) \twoheadrightarrow t$ implies that $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} \mu(X_j) \twoheadrightarrow t$ for some $\mu \in \{\mu_1, \dots, \mu_h\}$. But then $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} s_1?\dots?s_l \twoheadrightarrow p\mu$, and so $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} s \twoheadrightarrow p\mu$ for some $s \in \{s_1, \dots, s_l\}$. Hence, by Lemma 9 $t \sqsubseteq \mu(X_j)$ and $p\mu \sqsubseteq s$, and as $s_1 \dots s_l \equiv p\theta_1 \dots p\theta_l$ by definition then $p\mu \sqsubseteq s \equiv p\theta$ for some $\theta \in \{\theta_1, \dots, \theta_l\}$. But as $X_j \in \overline{X_j} \subseteq var(p)$ then $p\mu \sqsubseteq p\theta$ implies $\mu(X_j) \sqsubseteq \theta(X_j)$, hence $t \sqsubseteq \mu(X_j) \sqsubseteq \theta(X_j)$ and $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} \theta(X_j) \twoheadrightarrow t$ with a proof of the same size or smaller, by Proposition 1, and so $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} X_j(?\{\theta_1, \dots, \theta_l\}|_{var(r)}) \twoheadrightarrow t$ with a proof of the same size or smaller than the proof for $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} X_j[\overline{X_j/\mathit{project}_j(s_1?\dots?s_l)}] \twoheadrightarrow t$. But then we can apply Proposition 2 to get $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} r(?\{\theta_1, \dots, \theta_l\}|_{var(r)}) \twoheadrightarrow t$ to which we can apply the IH to get $\mathcal{P} \vdash_{\pi^\alpha CRWL} r(?\{\theta_1, \dots, \theta_l\}|_{var(r)}) \twoheadrightarrow t$. We can also apply the IH to each $pST(\mathcal{P}) \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t_i \equiv s_l \equiv p\theta_l$ (by definition of $s_1 \dots s_l$ there must exists some t_i corresponding to each s_l) and build the following proof:

$$\frac{\begin{array}{l} e \twoheadrightarrow s_1 \equiv p\theta_1 \\ \dots \\ e \twoheadrightarrow s_l \equiv p\theta_l \quad r(?\{\theta_1, \dots, \theta_l\}) \equiv r(?\{\theta_1, \dots, \theta_l\}|_{var(r)}) \twoheadrightarrow t \end{array}}{\mathcal{P} \vdash_{\pi^\alpha CRWL} f(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

□

Lemma 18. *For every $e \in Exp$ and $p \in CTerm$ linear, given $\theta \in CSubst_\bot$ such that $dom(\theta) \subseteq FV(p)$, if $p\theta \sqsubseteq |e|$ then $\exists \sigma \in Subst$ such that $dom(\sigma) = dom(\theta)$, $p\sigma \equiv e$ and $\theta \sqsubseteq \sigma$.*

Proof. By induction on the structure of $p\theta$:

Base cases

- $p\theta \equiv Y \in \mathcal{V}$: Then $p\theta \equiv Y \sqsubseteq |e|$ implies $Y \equiv |e|$ and so $Y \equiv e$. But then we can take $\sigma = \theta$ to get $p\sigma \equiv p\theta \equiv Y \equiv e$, $\theta \sqsubseteq \sigma$ as $\sigma \sqsubseteq \sigma$, and $dom(\sigma) = dom(\theta)$.
- $p\theta \equiv c \in CS^0$: Then $p\theta \equiv c \sqsubseteq |e|$ implies $c \equiv |e|$ and so $c \equiv e$. But then we can take $\sigma = \theta$ to get $p\sigma \equiv p\theta \equiv c \equiv e$, $\theta \sqsubseteq \sigma$ as $\sigma \sqsubseteq \sigma$, and $dom(\sigma) = dom(\theta)$.
- $p\theta \equiv \bot$: Then $p \equiv X \in \mathcal{V}$, and $\theta = [X/\ \bot]$, as $dom(\theta) \subseteq FV(p) = \{X\}$. But then we can choose $\sigma = [X/e]$ to get $p\sigma \equiv X[X/e] \equiv e$, $\theta = [X/\ \bot] \sqsubseteq [X/e] \equiv \sigma$, and $dom(\sigma) = \{X\} = dom(\theta)$.

Inductive steps

- $p\theta \equiv c(s_1, \ldots, s_n)$ with $p \equiv X \in \mathcal{V}$: Then $dom(\theta) \subseteq FV(p)$ implies $dom(\theta) = \{X\}$, so $\theta = [X/X\theta] = [X/p\theta]$. As $\theta \in CSubst_\bot, p \in CTerm$ then $p\theta \in CTerm_\bot$ and so $p\theta \sqsubseteq |e|$ implies $p\theta \sqsubseteq e$ by *b)*. But then we can choose $\sigma = [X/e]$ to get $p\sigma \equiv X[X/e] \equiv e$, $\theta = [X/p\theta] \sqsubseteq [X/e] \equiv \sigma$, and $dom(\sigma) = \{X\} = dom(\theta)$.
- $p\theta \equiv c(p_1\theta, \ldots, p_n\theta)$ with $p \equiv c(p_1, \ldots, p_n)$. Then $p\theta \equiv c(p_1\theta, \ldots, p_n\theta) \sqsubseteq |e|$ implies $|e| \equiv c(|e_1|, \ldots, |e_n|)$ for $e \equiv c(e_1, \ldots, e_n)$ such that $\forall i,\ p_i\theta \sqsubseteq |e_i|$. As p is linear and $dom(\theta) \subseteq FV(p)$ then if for every i we define $\theta_i = \theta|_{FV(p_i)}$ then $\theta = \theta_1 \uplus \ldots \uplus \theta_n$. But then for every i we have $p_i\theta_i \sqsubseteq |e_i|$ to which we can apply the IH to get $\exists \sigma_i \in Subst$ such that $p_i\sigma_i \equiv e_i$, $\theta_i \sqsubseteq \sigma_i$ and $dom(\sigma_i) = dom(\theta_i)$. But as p is linear then $\sigma = \sigma_1 \uplus \ldots \uplus \sigma_n$ is correctly defined and $p\sigma \equiv c(p_1\sigma_1, \ldots, p_n\sigma_n) \equiv c(e_1, \ldots, e_n) \equiv e$, $\theta \sqsubseteq \sigma$ and $dom(\sigma) = dom(\sigma_1) \cup \ldots \cup dom(\sigma_n) = dom(\theta_1) \cup \ldots \cup dom(\theta_n) = dom(\theta)$.

□

A typical approach for proving Lemma 8 would proceed by induction on the size of the starting $\pi^\alpha CRWL$-proof $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ of the hypothesis. Unfortunately it does not work, as we can see considering the case for the expression $f(e)$ with $f \in FS^{\mathcal{P}}$, and the following starting $\pi^\alpha CRWL$-proof, using a program rule $(f(p) \to r) \in \mathcal{P}$.

$$\frac{\begin{array}{l} e \twoheadrightarrow p\theta_1 \\ \ldots \\ e \twoheadrightarrow p\theta_m \qquad r?\{\theta_1, \ldots, \theta_m\} \twoheadrightarrow t \end{array}}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} f(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

The transformed rule corresponding to $(f(p) \to r)$ would be $f(Y) \to \mathit{if}\ \mathit{match}(Y)\ \mathit{then}\ r[\overline{X_i/\mathit{project}_i(Y)}]$, where $\overline{X_i} = var(p) \cap var(r)$, while the rules for *match* and each $\mathit{project}_i$ would be $\mathit{match}(p) \to \mathit{true}$, $\mathit{project}_i(p) \to X_i$.

We could then choose an arbitrary $\theta_j \in \{\theta_1, \ldots, \theta_m\}$ and apply the induction hypothesis over its corresponding $\pi^\alpha CRWL$-proof $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow p\theta_j$, thus getting $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \to^* e'_j$ such that $p\theta_j \sqsubseteq |e'_j|$. Then, as p is linear because it is in the left-hand side of a program rule, by Lemma 18 there must exists some $\sigma_j \in Subst$ such that $p\sigma_j \equiv e'_j$, so we can do

$$\begin{array}{l} \hat{\mathcal{P}} \uplus \mathcal{M} \vdash f(e) \to \mathit{if}\ \mathit{match}(e)\ \mathit{then}\ r[\overline{X_i/\mathit{project}_i(e)}] \\ \to^* \mathit{if}\ \mathit{match}(e'_j)\ \mathit{then}\ r[\overline{X_i/\mathit{project}_i(e)}] \\ \equiv \mathit{if}\ \mathit{match}(p\sigma_j)\ \mathit{then}\ r[\overline{X_i/\mathit{project}_i(e)}] \\ \to \mathit{if}\ \mathit{true}\ \mathit{then}\ r[\overline{X_i/\mathit{project}_i(e)}] \to r[\overline{X_i/\mathit{project}_i(e)}] \end{array}$$

The problem now is that we must find a way to apply the induction hypothesis over $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i/\mathit{project}_i(e)}] \twoheadrightarrow t$. Although we can in fact prove that this statement holds, as we will see later, the corresponding $\pi^\alpha CRWL$-proof cannot be granted to have in general a smaller or equal size than the proof for $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r?\{\theta_1, \ldots, \theta_m\} \twoheadrightarrow t$, as the computation for each $e \twoheadrightarrow p\theta_j$ statement might have

been replicated several times in the proof for $r[\overline{X_i/project_i(e)}] \twoheadrightarrow t$ because of the copies of e performed by the parameter passing substitution for the term rewriting step $f(e) \to \mathit{if}\ match(e)\ \mathit{then}\ r[\overline{X_i/project_i(e)}]$. Besides, another extra work is caused by the evaluation of the calls to each $project_i$ function, that does not have to be performed in $r?\{\theta_1, \ldots, \theta_m\} \twoheadrightarrow t$.

This can be seen for example in the sample use of the transformation at the beginning of Section 4.3.1, where the first term rewriting step makes several copies of $c(0)\ ?\ c(1)$. There $r?\{\theta_1, \ldots, \theta_m\}$ would be $d(0\ ?\ 1, 0\ ?\ 1)$ and $r[\overline{X_i/project_i(e)}]$ would be $d(project(c(0)\ ?\ c(1)), project(c(0)\ ?\ c(1)))$: although it is clear that both expressions can be reduced by $\pi^{\alpha}CRWL$ to e.g., the value $d(0, 1)$, it is also clear that the $\pi^{\alpha}CRWL$-proof for the latter expression will be bigger than the one for the former.

The conclusion then is that we must look for another well-founded relation upon our inductive proof could be based on. To do that we must first introduce the following notions, which allow us to talk more directly about the building blocks of $\pi^{\alpha}CRWL$-proofs.

Definition 10. *Given a $\pi^{\alpha}CRWL$-proof Δ for a $\pi^{\alpha}CRWL$-statement $e \twoheadrightarrow t$.*

- *Π^{Δ} denotes the multiset of $\pi^{\alpha}CRWL$-statements that compose Δ, including $e \twoheadrightarrow t$. We will sometimes use $\Pi^{e \twoheadrightarrow t}$ when Δ is implicit.*
- *π is the metavariable we will use to refer to a $\pi^{\alpha}CRWL$-statement.*
- *Δ_{π} refers to the subproof for some premise π of Δ, when Δ is implicit.*

We can now use these notions to define the well-founded relation we are looking for.

Definition 11. *For any proof Δ for $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$ we define $sa\mathcal{P}(\Delta_{e \twoheadrightarrow t})$—statements after $\mathcal{P}$—as:*

$$sa\mathcal{P}(\Delta_{e \twoheadrightarrow t}) = \{e' \twoheadrightarrow t' \mid (e' \twoheadrightarrow t') \in \Pi^{e \twoheadrightarrow t} \wedge e' \equiv f(\overline{a})\ \mathit{for\ some}\ f \in FS^{\mathcal{P}}\}$$

Note that $sa\mathcal{P}(e \twoheadrightarrow t)$ is a set, not a multiset. For any $\pi^{\alpha}CRWL$-proof Δ by $size(\Delta)$ we denote the number of rules of the calculus used.
We define the relation $\lessdot$ over pairs of $\pi^{\alpha}CRWL$-proofs Δ_1, Δ_2, by

$$\Delta_1 \lessdot \Delta_2\ \mathit{iff}\ \begin{array}{c} sa\mathcal{P}(\Delta_1) \subset sa\mathcal{P}(\Delta_2) \\ \mathit{or} \\ sa\mathcal{P}(\Delta_1) = sa\mathcal{P}(\Delta_2)\ \mathit{and}\ size(\Delta_1) < size(\Delta_2) \end{array}$$

For the relation $\lessdot$ to be useful in inductions it must be well-founded, the subject of the following result.

Lemma 19. *For any program $\mathcal{P} \uplus \mathcal{M}$ the relation $\lessdot$ over pairs of $\pi^{\alpha}CRWL$-proofs is well-founded.*

Proof. It is enough to prove that there is no infinite descending chain $\ldots \lessdot \Delta_3 \lessdot \Delta_2 \lessdot \Delta_1$ [7]. And this chain does not exists because $\pi^{\alpha}CRWL$-proofs are finite and therefore for any proof Δ we have that $sa\mathcal{P}(\Delta)$ is finite—thus there are finitely many proper subsets of it—and $size(\Delta)$ is a natural number. □

Now we will prove Lemma 8 by induction over $\lessdot$ applied to $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$. To do that we still need some auxiliary results. Next result is pretty technical, but just states the existence of a class of $\pi^{\alpha}CRWL$-proofs that do not have useless loops inside.

Lemma 20. *For any $\pi^{\alpha}CRWL$-statement $e \twoheadrightarrow t$ which holds exists some $\pi^{\alpha}CRWL$-proof Δ such that $\forall e' \twoheadrightarrow t' \in \Pi^{\Delta}$, $e' \twoheadrightarrow t'$ is not a premise, neither directly nor indirectly, of itself in Δ.*

Proof. As $e \twoheadrightarrow t$ holds we may assume some $\pi^{\alpha}CRWL$-proof Δ for it. If no $e' \twoheadrightarrow t' \in \Pi^{\Delta}$ is premise of itself then we are done. Otherwise as any $\pi^{\alpha}CRWL$-proof is finite then taking the subproof corresponding to some $e' \twoheadrightarrow t'$ which is premise of itself there must be some $e' \twoheadrightarrow t'$ which does not have $e' \twoheadrightarrow t'$ as its premise in its corresponding proof. Then we can use that proof to replace the subproof for $e' \twoheadrightarrow t'$, as the proof is finite this process ends, because each time the number of sentences premises of itself decreases. □

The following result shows how $\lessdot$ in some sense extends the principle of induction over the structure of a $\pi^{\alpha}CRWL$-proof.

Lemma 21. *Under any program $\mathcal{P} \uplus \mathcal{M}$, for any a proof Δ for $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^{\alpha}CRWL} e \twoheadrightarrow t$ and any $\pi \in \Pi^{e \twoheadrightarrow t}$.*

- $sa\mathcal{P}(\Delta_\pi) \subseteq sa\mathcal{P}(\Delta_{e \twoheadrightarrow t})$.
- *If* $\pi \not\equiv e \twoheadrightarrow t$ *then* $\Delta_\pi \lessdot \Delta_{e \twoheadrightarrow t}$*: proper direct or indirect premises are smaller w.r.t.* $\lessdot$.

Proof. First part holds because $\pi \in \Pi^{e \twoheadrightarrow t}$ implies $\Pi^\pi \subseteq \Pi^{e \twoheadrightarrow t}$, but then $sa\mathcal{P}(\Delta_\pi) \subseteq sa\mathcal{P}(\Delta_{e \twoheadrightarrow t})$ by definition. Concerning the second, by the first part we only have to prove that $size(\Delta_\pi) < size(\Delta_{e \twoheadrightarrow t})$, which is a consequence of $\pi \in \Pi^{e \twoheadrightarrow t}$, $\pi \not\equiv e \twoheadrightarrow t$. □

From now on we will assume that every $\pi^\alpha CRWL$-proof that we consider fulfills the requirements of Lemma 20. Now we are ready to prove Lemma 8.

Proof for Lemma 8. We assume we start with a proof for $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$ which fulfils the conditions granted by Lemma 20, and proceed by induction over $\lessdot$ applied over $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow t$. Let us do a case distinction over the rule applied at the root of the proof:

B Then $t \equiv \bot$ and $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \to^0 e$ for which $\bot \sqsubseteq |e|$ holds.

RR Then $e \equiv X \equiv t$ and $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \to^0 e$ for which $X \sqsubseteq X \equiv |e|$ holds.

DC If $e \equiv c \in CS^0$ then $t \equiv c$ and so $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \to^0 e$ for which $c \sqsubseteq c \equiv |e|$ holds. Otherwise $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \equiv c(e_1, \ldots, e_n) \twoheadrightarrow c(t_1, \ldots, t_n) \equiv t$. But then by Lemma 21 we can apply the IH over each $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e_i \twoheadrightarrow t_i$ of the starting proof, thus getting $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e_i \to^* e'_i$ such that $t_i \sqsubseteq |e'_i|$. But then $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \equiv c(e_1, \ldots, e_n) \to^* c(e'_1, \ldots, e'_n)$, for which $t \equiv c(t_1, \ldots, t_n) \sqsubseteq c(|e'_1|, \ldots, |e'_n|) \equiv |c(e'_1, \ldots, e'_n)|$.

POR$^\alpha$ Then we have $f(\bar{e}) \twoheadrightarrow t$. In case $t \equiv \bot$ lemma holds trivially for $f(\bar{e}) \to^0 f(\bar{e})$ with $t \equiv \bot \sqsubseteq \bot \equiv |f(\bar{e})|$. Otherwise we proceed by a case distinct over f.
If $f \in FS^{\mathcal{P}}$ for the sake of sake of simplicity we will consider $f \in FS^1$—the proof can be easily extended to functions with zero or more than one arguments, due to left linearity of program rules. Assume the rule used was $(f(p) \to r) \in \mathcal{P}$, so we have the following starting proof.

$$\frac{\begin{array}{l} e \twoheadrightarrow p\theta_1 \\ \ldots \\ e \twoheadrightarrow p\theta_m \qquad r?\{\theta_1, \ldots, \theta_m\} \twoheadrightarrow t \end{array}}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} f(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

The transformed rule corresponding to $(f(p) \to r)$ would be $f(Y) \to \mathit{if}\ \mathit{match}(Y)\ \mathit{then}\ r[\overline{X_i / \mathit{project}_i(Y)}]$, where $\overline{X_i} = var(p) \cap var(r)$, while the rules for *match* and each $\mathit{project}_i$ would be $\mathit{match}(p) \to \mathit{true}$, $\mathit{project}_i(p) \to X_i$. Then by Lemma 21 we can apply the IH over any $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \twoheadrightarrow p\theta_j$, so we pick an arbitrary $\theta_j \in \{\theta_1, \ldots, \theta_m\}$ thus getting $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e \to^* e'_j$ such that $p\theta_j \sqsubseteq |e'_j|$. Then, as p is linear because it is in the left-hand side of a program rule, by Lemma 18 there must exists some $\sigma_j \in \mathit{Subst}$ such that $p\sigma_j \equiv e'_j$, so we can do

$$\begin{array}{l} \hat{\mathcal{P}} \uplus \mathcal{M} \vdash f(e) \to \mathit{if}\ \mathit{match}(e)\ \mathit{then}\ r[\overline{X_i / \mathit{project}_i(e)}] \\ \to^* \mathit{if}\ \mathit{match}(e'_j)\ \mathit{then}\ r[\overline{X_i / \mathit{project}_i(e)}] \\ \equiv \mathit{if}\ \mathit{match}(p\sigma_j)\ \mathit{then}\ r[\overline{X_i / \mathit{project}_i(e)}] \\ \to \mathit{if}\ \mathit{true}\ \mathit{then}\ r[\overline{X_i / \mathit{project}_i(e)}] \to r[\overline{X_i / \mathit{project}_i(e)}] \end{array}$$

Now, in order to enable the application of the induction hypothesis over $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i / \mathit{project}_i(e)}] \twoheadrightarrow t$ we will prove that this statement holds, and also do some calculations with $sa\mathcal{P}(\_)$. By Lemma 20 we can assume a $\pi^\alpha CRWL$-proof Δ for $f(e) \twoheadrightarrow t$ such that it is not a premise of itself, and so $sa\mathcal{P}(\Delta_{f(e) \twoheadrightarrow t}) = \{f(e) \twoheadrightarrow t\} \uplus \mathcal{S}$, where $\mathcal{S} = (\bigcup_{j \in \{1,\ldots,m\}} sa\mathcal{P}(\Delta_{e \twoheadrightarrow p\theta_j})) \cup sa\mathcal{P}(\Delta_{r?\{\theta_1,\ldots,\theta_m\} \twoheadrightarrow t})$. Now, if we could prove that there is a proof Δ' for $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i / \mathit{project}_i(e)}] \twoheadrightarrow t$ such that $sa\mathcal{P}(\Delta') \subseteq \mathcal{S}$, then we would have $sa\mathcal{P}(\Delta') \subseteq \mathcal{S} \subset sa\mathcal{P}(\Delta_{f(e) \twoheadrightarrow t})$ so we can apply the induction hypothesis to get a derivation $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash r[\overline{X_i / \mathit{project}_i(e)}] \to^* e'$ such that $t \sqsubseteq |e'|$.

Therefore, all that is left is proving that such a proof Δ' exists. To do that we proceed by induction on the structure of r—taking into account that $r \in \mathit{Exp}$ as it is the right-hand side of a program rule—and proving that, from $\Delta_{r?\{\theta_1,\ldots,\theta_m\} \twoheadrightarrow t}$, we can build a proof Δ' for $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i / \mathit{project}_i(e)}] \twoheadrightarrow t$ under the conditions above. Regarding the base cases.

- If $r \equiv Y \in \mathcal{V}$: in order to make the proof more solid, we will consider here the more general case where $f \in FS^n$ for any $n \geq 0$, so the starting statement is $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} f(\overline{e}) \twoheadrightarrow t$, the rule used is $(f(\overline{p}) \to r) \in \mathcal{P}$ and the proof Δ' is being built for the statement $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_{ij}/project_{ij}(e_i)}] \twoheadrightarrow t$. Then $\{Y\} = var(r) \subseteq var(\overline{p})$, because of the absence of extra variables, therefore $\exists p_i \in \overline{p}.Y \in var(p_i)$. But as $\overline{X_{ij}} = var(p_i) \cap var(r)$ we get $\exists X_{ij} \in \overline{X_{ij}}.Y \equiv X_{ij}$, as a consequence, and so $r[\overline{X_{ij}/project_{ij}(e_i)}] \equiv Y[\overline{X_{ij}/project_{ij}(e_i)}] \equiv project_{ij}(e_i)$.

  Going back to the case where $f \in FS^1$, that implies that $r[\overline{X_i/project_i(e)}] \equiv project_Y(e)$. We will now use the hypothesis $\Delta_{r?\{\theta_1,\ldots,\theta_m\} \twoheadrightarrow t}$ proof to find our Δ'. We have two possibilities.

  1. $\forall \theta \in \{\theta_1, \ldots, \theta_m\}.Y \in dom(\theta)$: Then $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r?\{\theta_1, \ldots, \theta_m\} \equiv \theta_1(Y) \,?\, \ldots \,?\, \theta_m(Y) \twoheadrightarrow t$. But that implies that $\exists \theta_j \in \{\theta_1, \ldots, \theta_m\}.\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} \theta_j(Y) \twoheadrightarrow t$, so we can build Δ' as.

  $$\dfrac{\dfrac{hypothesis}{e \twoheadrightarrow p\theta_j} \quad \theta_j(Y) \twoheadrightarrow t}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i/project_i(e)}] \equiv project_Y(e) \twoheadrightarrow t}\ \mathbf{POR}^\alpha$$

  But $\theta_j \in CSubst_\bot$ implies $sa\mathcal{P}(\Delta'_{\theta_j(Y) \twoheadrightarrow t}) = \emptyset$. Besides, we have defined $\Delta'_{e \twoheadrightarrow p\theta_j}$ as $\Delta_{e \twoheadrightarrow p\theta_j}$, and $sa\mathcal{P}(\Delta_{e \twoheadrightarrow p\theta_j}) \subseteq \mathcal{S}$ by definition of $\mathcal{S}$, therefore $sa\mathcal{P}(\Delta') = sa\mathcal{P}(\Delta'_{r[\overline{X_i/project_i(e)}] \twoheadrightarrow t}) = sa\mathcal{P}(\Delta'_{\theta_j(Y) \twoheadrightarrow t}) \cup sa\mathcal{P}(\Delta'_{e \twoheadrightarrow p\theta_j}) \subseteq \mathcal{S}$.

  2. $\exists \theta_j \in \{\theta_1, \ldots, \theta_m\}.Y \notin dom(\theta_j)$: Then $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r?\{\theta_1, \ldots, \theta_m\} \equiv Y \,?\, \rho_1(Y) \,?\, \ldots \,?\, \rho_l(Y) \twoheadrightarrow t$, for $\{\rho_1, \ldots, \rho_l\} \subseteq \{\theta_1, \ldots, \theta_m\}$. But then, either $\exists \rho_k \in \{\rho_1, \ldots, \rho_l\}.\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} \rho_k(Y) \twoheadrightarrow t$ so we can do like in the previous case; or $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} Y \twoheadrightarrow t$. As we discarted $t \equiv \bot$ at the beginning of the case for $\mathbf{POR}^\alpha$, that implies $t \equiv Y$, so we can build Δ' as.

  $$\dfrac{\dfrac{hypothesis}{e \twoheadrightarrow p\theta_j} \quad \dfrac{}{\theta_j(Y) \equiv Y \twoheadrightarrow Y}\ \mathbf{RR}}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i/project_i(e)}] \equiv project_Y(e) \twoheadrightarrow Y \equiv t}\ \mathbf{POR}^\alpha$$

  by using that $\theta_j \in \{\theta_1, \ldots, \theta_m\}.Y \notin dom(\theta_j)$ above. Like in the previuos case $sa\mathcal{P}(\Delta'_{Y \twoheadrightarrow Y}) = \emptyset$, and we can do the same computations with $sa\mathcal{P}(\_)$ to conclude $sa\mathcal{P}(\Delta') \subseteq \mathcal{S}$.

- If $r \equiv c \in CS^0$: Then $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r?\{\theta_1, \ldots, \theta_m\} \equiv c \twoheadrightarrow t$ implies $t \equiv c$, because $t \not\equiv \bot$, so we can build Δ' as.

  $$\dfrac{}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i/project_i(e)}] \equiv c \twoheadrightarrow c \equiv t}\ \mathbf{DC}$$

  with $sa\mathcal{P}(\Delta') = sa\mathcal{P}(\Delta'_{r[\overline{X_i/project_i(e)}] \twoheadrightarrow t}) = sa\mathcal{P}(\Delta'_{c \twoheadrightarrow c}) = \emptyset \subseteq \mathcal{S}$.

- The case for $r \equiv g \in FS^0$ is included in the general case for $g \in FS$ below.

Concerning the inductive steps.

- If $r \equiv c(a_1, \ldots, a_l)$ for $c \in CS^l$: As $t \not\equiv \bot$ then the hypothesis must be $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r?\{\theta_1, \ldots, \theta_m\} \equiv c(a_1?\{\theta_1, \ldots, \theta_m\}, \ldots, a_l?\{\theta_1, \ldots, \theta_m\}) \twoheadrightarrow c(t_1, \ldots, t_l) \equiv t$, where for each a_k we have $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} a_k?\{\theta_1, \ldots, \theta_m\} \twoheadrightarrow t_k$. But then by IH over each a_k we get a proof $\Delta'_{a_k[\overline{X_i/project_i(e)}] \twoheadrightarrow t_k}$, and we can combine all these proofs together to build Δ' as a proof for the statement

  $$\begin{array}{l}\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r[\overline{X_i/project_i(e)}] \\ \equiv c(a_1[\overline{X_i/project_i(e)}], \ldots, a_l[\overline{X_i/project_i(e)}]) \twoheadrightarrow c(t_1, \ldots, t_l) \equiv t\end{array}$$

  by perfoming a $\mathbf{DC}$ step at the root of Δ' and using each $\Delta'_{a_k[\overline{X_i/project_i(e)}] \twoheadrightarrow t_k}$ as hypothesis. But then $sa\mathcal{P}(\Delta') = \bigcup_{k \in \{1,\ldots,l\}} sa\mathcal{P}(\Delta'_{a_k[\overline{X_i/project_i(e)}] \twoheadrightarrow t_k}) \subseteq \mathcal{S}$, as for each a_k we have $sa\mathcal{P}(\Delta'_{a_k[\overline{X_i/project_i(e)}] \twoheadrightarrow t_k}) \subseteq \mathcal{S}$ by IH.

- If $r \equiv g(a_1, \ldots, a_l)$ for $g \in FS^l$: As $t \not\equiv \bot$ then the $\pi^\alpha CRWL$ rule used at the root of $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} r?\{\theta_1, \ldots, \theta_m\} \equiv g(a_1, \ldots, a_l)?\{\theta_1, \ldots, \theta_m\} \twoheadrightarrow t$ must be $\mathbf{POR}^\alpha$, let $r'\mu$ be the right-hand side of the rule and the parameter passing substitution used in that step. We can proceed like in the previous case, applying the IH over each $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL}$

$a_k?\{\theta_1,\ldots,\theta_m\} \twoheadrightarrow s_j$ getting a proof $\Delta'_{a_k[\overline{X_i/project_i(e)}]\twoheadrightarrow s_j}$, and then build Δ' by using $\mathbf{POR}^\alpha$ at the root, and $\Delta_{r'\mu\twoheadrightarrow t}$ as the last hypothesis. On one hand we have $sa\mathcal{P}(\Delta'_{a_k[\overline{X_i/project_i(e)}]\twoheadrightarrow s_j}) \subseteq \mathcal{S}$ for each of these premises by IH. On the other hand

$$\begin{array}{l} sa\mathcal{P}(\Delta'_{r'\mu\twoheadrightarrow t}) = sa\mathcal{P}(\Delta_{r'\mu\twoheadrightarrow t}) \\ \subseteq sa\mathcal{P}(\Delta_{g(a_1,\ldots,a_l)?\{\theta_1,\ldots,\theta_m\}\twoheadrightarrow t}) \\ = sa\mathcal{P}(\Delta_{r?\{\theta_1,\ldots,\theta_m\}\twoheadrightarrow t}) \subseteq \mathcal{S} \end{array}$$

therefore $sa\mathcal{P}(\Delta') \subseteq \mathcal{S}$.

If $e \equiv match(e_1,\ldots,e_n)$ for some of these auxiliary *match* functions in $\mathcal{M}$, with rule $match(p_1,\ldots,p_n) \to true$, then as $t \not\equiv \bot$ we have $t \equiv true$ with:

$$\frac{e_1 \twoheadrightarrow p_1\theta_1 \quad \ldots \quad e_n \twoheadrightarrow p_n\theta_n \quad true \twoheadrightarrow true}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \equiv match(e_1,\ldots,e_n) \twoheadrightarrow true}\ \mathbf{POR}$$

There could be more evaluations for each e_i but those are useless because *true* is ground. Then by Lemma 21 we can apply the IH over each $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e_i \twoheadrightarrow p_i\theta$ so we get $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e_i \to^* e'_i$ such that $p_i\theta_i \sqsubseteq |e'_i|$. But then by Lemma 18 there must exist some $\sigma_i \in Subst$ such that $p_i\sigma_i \equiv e'_i$, and we can do $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash match(e_1,\ldots,e_n) \to^* match(e'_1,\ldots,e'_n) \equiv match(p_1\sigma_1,\ldots,p_n\sigma_n) \to true$, and $true \sqsubseteq true \equiv |true|$.

If $e \equiv project(e_1)$ for some of these auxiliary functions, with rule $project(p) \to X$, then as $t \not\equiv \bot$ we have:

$$\frac{\begin{array}{l} e_1 \twoheadrightarrow p\theta_1 \\ \ldots \\ e_1 \twoheadrightarrow p\theta_m \qquad X(?\{\theta_1,\ldots,\theta_m\}) \twoheadrightarrow t \end{array}}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \equiv project(e_1) \twoheadrightarrow t}\ \mathbf{POR}$$

Then there must exist some $\theta_j \in \{\theta_1,\ldots,\theta_m\}$ such that $X\theta_j \twoheadrightarrow t$, hence $t \sqsubseteq \theta_j(X)$ by Lemma 9. But then by Lemma 21 we can apply the IH over $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e_1 \twoheadrightarrow p\theta_j$ so we get $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e_1 \to^* e'_j$ such that $p\theta_j \sqsubseteq |e'_j|$, to which we can apply Lemma 18 to get some $\sigma_j \in Subst$ such that $p\sigma_j \equiv e'_j$ and $\theta_j \sqsubseteq \sigma_j$. Therefore $t \sqsubseteq \theta_j(X) \sqsubseteq \sigma_j(X)$, so it is trivial to check that then $t \sqsubseteq |\sigma_j(X)|$ (by induction on the structure of t), and we can do $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash match(e_1) \to^* match(e'_j) \equiv match(p\sigma_j) \to \sigma_j(X)$.

If $e \equiv if\ e_1\ then\ e_2$ then as $t \not\equiv \bot$ we have:

$$\frac{\begin{array}{lll} & e_2 \twoheadrightarrow t_1 & \\ & \ldots & \\ e_1 \twoheadrightarrow true & e_2 \twoheadrightarrow t_m & t_1?\ldots?t_m \twoheadrightarrow t \end{array}}{\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e \equiv if\ e_1\ then\ e_2 \twoheadrightarrow t}\ \mathbf{POR}$$

There could be more evaluations for e_1 but those are useless because *true* is ground. But then by Lemma 21 we can apply the IH over $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e_1 \twoheadrightarrow true$ so we get $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e_1 \to^* e'_1$ such that $true \sqsubseteq |e'_1|$. Then we can apply Lemma 18 to get some $\sigma_1 \in Subst$ such that $true \equiv true\sigma_1 \equiv e'_1$, so we can do $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash if\ e_1\ then\ e_2 \to^* if\ e'_1\ then\ e_2 \equiv if\ true\ then\ e_2 \to e_2$. Besides $t_1?\ldots?t_m \twoheadrightarrow t$ implies $t_j \twoheadrightarrow t$ for some $t_j \in \{t_1,\ldots,t_m\}$ such that $t \sqsubseteq t_j$, by Lemma 9. But then by Lemma 21 we can apply the IH over $\mathcal{P} \uplus \mathcal{M} \vdash_{\pi^\alpha CRWL} e_2 \twoheadrightarrow t_j$ so we get $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e_2 \to^* e'$ such that $t \sqsubseteq t_j \sqsubseteq |e'|$.

If $e \equiv e_1\ ?\ e_2$ then as $t \not\equiv \bot$ we have $e_i \twoheadrightarrow t$ for some $i \in \{1,2\}$ with a proof to which we can apply the IH by Lemma 21 to get $\hat{\mathcal{P}} \uplus \mathcal{M} \vdash e_1\ ?\ e_2 \to e_i \to^* e'$ such that $t \sqsubseteq |e'|$.

□

A.5 For Section 5.1

Theorem 10 (Compositionality of $CRWL^\sigma_{\pi^\alpha}$ and $CRWL^\sigma_{\pi^\beta}$)**.** *For any program, $\mathcal{C} \in Cntxt$ and $e, e' \in Exp_\perp$:*

$$\begin{array}{l} [\![\mathcal{C}[e]]\!]^{s\alpha p} = \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![e]\!]^{s\alpha p}} [\![\mathcal{C}[t_1\ ?\ \ldots\ ?\ t_n]]\!]^{s\alpha p} \\ [\![\mathcal{C}[e]]\!]^{s\beta p} = \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![e]\!]^{s\beta p}} [\![\mathcal{C}[t_1\ ?\ \ldots\ ?\ t_n]]\!]^{s\beta p} \end{array}$$

for any arrangement of the elements of $\{t_1, \ldots, t_n\}$ *in* $t_1 \ ? \ \ldots \ ? \ t_n$. *As a consequence:*

$$[\![e]\!]^{s\circ p} = [\![e']\!]^{s\circ p} \text{ iff } \forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!]^{s\circ p} = [\![\mathcal{C}[e']]\!]^{s\circ p}$$
$$[\![e]\!]^{s\beta p} = [\![e']\!]^{s\beta p} \text{ iff } \forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!]^{s\beta p} = [\![\mathcal{C}[e']]\!]^{s\beta p}$$

Proof. A straightforward modification of the proof for Theorem 1. □

Proposition 7 (Compositionality of $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$ for singular contexts)**.** *For any program, singular context* $s\mathcal{C}$ *and* $e \in Exp_{\perp}$*:*

$$[\![s\mathcal{C}[e]]\!]^{s\circ p} = \bigcup_{t \in [\![e]\!]^{s\circ p}} [\![s\mathcal{C}[t]]\!]^{s\circ p}$$
$$[\![s\mathcal{C}[e]]\!]^{s\beta p} = \bigcup_{t \in [\![e]\!]^{s\beta p}} [\![s\mathcal{C}[t]]\!]^{s\beta p}$$

Proof. Again, a straightforward modification of the proof for Theorem 1. □

Proposition 8 (Bubbling for singular contexts in $CRWL^{\sigma}_{\pi^{\alpha}}$ and $CRWL^{\sigma}_{\pi^{\beta}}$)**.** *For any program, singular context* $s\mathcal{C}$ *and* $e_1, e_2 \in Exp_{\perp}$*:*

$$[\![s\mathcal{C}[e1 \ ? \ e_2]]\!]^{s\circ p} = [\![s\mathcal{C}[e_1] \ ? \ s\mathcal{C}[e_2]]\!]^{s\circ p}$$
$$[\![s\mathcal{C}[e1 \ ? \ e_2]]\!]^{s\beta p} = [\![s\mathcal{C}[e_1] \ ? \ s\mathcal{C}[e_2]]\!]^{s\beta p}$$

Proof. A straightforward adaptation of the proof for Proposition 4. □