Generators: Detailed Proofs*

Adrián Riesco and Juan Rodríguez-Hortalá

Technical Report 07/12

Departamento de Sistemas Informáticos y Computación,
Universidad Complutense de Madrid

March, 2012

*Research partially supported by the Spanish projects *DESAFIOS10* (TIN2009-14599-C03-01), *FAST-STAMP* (TIN2008-06622-C03-01/TIN), *PROMETIDOS-CM* (S2009/TIC-1465), and *GPD-UCM* (UCM-BSCH-GR58/08-910502).

Abstract

Narrowing is a procedure that was first studied in the context of equational E-unification, and that later has been used in a wide range of applications. The classic completeness result due to Hullot states that any term rewriting derivation starting from an instance of an expression can be 'lifted' to a narrowing derivation, whenever the substitution employed is normalized. In this paper we adapt the generator-based extra-variables-elimination transformation used in functional-logic programming to overcome that limitation, so we are able to lift term rewriting derivations starting from arbitrary instances of expressions. The proposed technique is limited to constructor systems and to derivations reaching a ground expression. We also present a Maude-based implementation of the technique, using natural rewriting for the on-demand evaluation strategy.

Keywords: Term Rewriting, Constructor Systems, Lifting, Generators, Maude.

Contents

1 Introduction

Narrowing [2] is a procedure that was first studied in the context of equational E-unification, and that later has been used in a wide range of applications [15, 9]. Narrowing can be described as a modification of term rewriting in which matching is replaced by unification so, in a derivation starting from a goal expression, it is able to deduce the instantiation of the variables of the goal expression that is needed for the computation to progress. The key result for the completeness of narrowing w.r.t. term rewriting is *Hullot's lifting lemma* [11], which states that any term rewriting derivation $e_1\theta \rightarrow^* e_2$ can be *lifted* into a narrowing derivation $e_1 \leadsto^*_\sigma e_3$ such that e_3 and σ are more general than e_2 and θ—w.r.t. to the usual instantiation preorder [3], and for the variables involved in the derivations—, provided that the starting substitution θ is normalized [16]. This latter condition is essential, so it is fairly easy to break Hullot's lifting lemma by dropping it: e.g. under the term rewriting system (TRS) $\{f(0,1) \rightarrow 2, coin \rightarrow 0, coin \rightarrow 1\}$ the term rewriting derivation $f(X,X)[X/coin] \rightarrow^* 2$ cannot be lifted by any narrowing derivation. Several variants and extensions of narrowing have been developed in order to improve that result under certain assumptions or for particular classes of term rewriting systems [16, 15, 7].

In this paper we show how to adapt the generator-based extra variable elimination transformation used in functional-logic programming (FLP) to drop the normalization condition required by Hullot's lifting lemma. The proposed technique is devised for constructor systems (CS's) with extra variables, and it is limited to derivations reaching a ground expression. To test the feasibility of this approach, we have also developed a prototype in Maude [5], relying on the natural rewriting on-demand strategy [8] to obtain an effective operational procedure.

2 Preliminaries and formal setting

We mostly use the notation from [2], with some additions from [13]. We consider a first order signature $\Sigma = CS \uplus FS$, where CS and FS are two disjoint set of *constructor* and defined *function* symbols respectively, all them with associated arity. 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. 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 $e_1, \ldots, e_n \in Exp$. The set $CTerm$ of *constructed terms* (or *c-terms*) is defined like Exp, but with h restricted to CS^n (so $CTerm \subseteq Exp$). We will write $e, e', \ldots$ for expressions and $t, s, \ldots$ for c-terms. We say that an expression e is *ground* iff no variable appears in e. We will frequently use *one-hole contexts*, defined as $Cntxt \ni \mathcal{C} ::= [\,] \mid h(e_1, \ldots, \mathcal{C}, \ldots, e_n)$. We also consider the extended signature $\Sigma_\bot = \Sigma \cup \{\bot\}$, where $\bot$ is a new 0-arity constructor symbol that does not appear in programs, and that stands for the undefined value. Over this signature we define the sets $Exp_\bot$ and $CTerm_\bot$ of *partial* expressions and c-terms, respectively. The intended meaning is that Exp and $Exp_\bot$ stand for evaluable expressions, i.e., expressions that can contain function symbols, while $CTerm$ and $CTerm_\bot$ stand for data terms representing total and partial values, respectively. 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 $e, e' \in Exp_\bot, \mathcal{C} \in Cntxt$. The *shell* $|e|$ of an expression e represents the outer constructed part of e and is defined as: $|X| = X$; $|c(e_1, \ldots, e_n)| = c(|e_1|, \ldots, |e_n|)$; $|f(e_1, \ldots, e_n)| = \bot$. It is trivial to check that for any expression e we have $|e| \in CTerm_\bot$, that any total expression is maximal w.r.t. $\sqsubseteq$, and that as as consequence if t is total then $t \sqsubseteq |e|$ implies $t = e$.

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 $e\theta$ to apply of θ to e, and $\theta\theta'$ for the composition, defined by $X(\theta\theta') = (X\theta)\theta'$. The domain and variable range of θ are defined as $dom(\theta) = \{X \in \mathcal{V} \mid X\theta \neq X\}$ and $vran(\theta) = \bigcup_{X \in dom(\theta)} var(X\theta)$. By $[X_1/e_1, \ldots, X_n/e_n]$ we denote a substitution σ such that $dom(\sigma) = \{X_1, \ldots, X_n\}$ and $\forall i.\sigma(X_i) = e_i$. 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 θ_i; $(\theta_0 \uplus \theta_1)(X) = X$ otherwise. Given $W \subseteq \mathcal{V}$ we write $\theta|_W$ for the restriction of θ to W, i.e. $(\theta|_W)(X) = \theta(X)$ if $X \in W$, and $(\theta|_W)(X) = X$ otherwise; we use $\theta|_{\setminus D}$ as a shortcut for $\theta|_{(\mathcal{V} \setminus D)}$. *C-substitutions* $\theta \in CSubst$ verify that $X\theta \in CTerm$ for all $X \in dom(\theta)$. We say a substitution σ is ground iff $vran(\sigma) = \emptyset$, i.e. $\forall X \in dom(\sigma)$ we have that $\sigma(X)$ is ground. The sets $Subst_\bot$ and $CSubst_\bot$ of partial substitutions and partial c-substitutions are the sets of finite mappings from variables to partial expressions and partial c-terms, respectively.

A constructor-based term rewriting system or just *constructor system* or *program* $\mathcal{P}$ (CS) is a set of c-rewrite rules of the form $f(\overline{t}) \rightarrow r$ where $f \in FS^n$, $r \in Exp$ and $\overline{t}$ is a linear n-tuple of c-terms, where linearity means that variables occur only once in $\overline{t}$. Notice that we allow r to contain so called *extra*

E	$se \twoheadrightarrow \emptyset$	
RR	$\{X\} \twoheadrightarrow \{X\}$	if $X \in \mathcal{V}$
DC	$\dfrac{se_1 \twoheadrightarrow st_1 \ \ldots \ se_n \twoheadrightarrow st_n}{\{c(se_1,\ldots,se_n)\} \twoheadrightarrow \{c(st_1,\ldots,st_n)\}}$	if $c \in CS$
More	$\dfrac{se \twoheadrightarrow st_1 \ \ldots se \twoheadrightarrow st_n}{se \twoheadrightarrow st_1 \cup \ldots \cup st_n}$	
Less	$\dfrac{\{esa_1\} \twoheadrightarrow st_1 \ \ldots \ \{esa_m\} \twoheadrightarrow st_m}{\{ese_1,\ldots,ese_n\} \twoheadrightarrow st_1 \cup \ldots \cup st_m}$	if $n \geq 2, m > 0$, for any $\{esa_1,\ldots,esa_m\} \subseteq \{ese_1,\ldots,ese_n\}$
ROR	$\dfrac{se_1 \twoheadrightarrow \widetilde{p_1}\theta \ \ldots \ se_n \twoheadrightarrow \widetilde{p_n}\theta \ \ \widetilde{r}\theta \twoheadrightarrow st}{\{f(se_1,\ldots,se_n)\} \twoheadrightarrow st}$	if $(f(p_1,\ldots,p_n) \to r) \in \mathcal{P}$ $\theta \in SCSubst$

Figure 1: A proof calculus for constructor systems

variables, i.e., variables not occurring in $f(\bar{t})$. 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. We assume that every CS contains the rules $\mathcal{Q} = \{X \ ? \ Y \to X, X \ ? \ Y \to Y\}$, defining the behavior of $? \in FS^2$, used in infix mode, and that those are the only rules for ?. Besides, ? is right-associative so $e_1 \ ? \ e_2 \ ? \ e_3$ is equivalent to $e_1 \ ? \ (e_2 \ ? \ e_3)$. For the sake of conciseness we will often omit these rules when presenting a CS. A consequence of this is that we only consider non-confluent programs. Given a TRS $\mathcal{P}$, 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 $\to_{\mathcal{P}}$. We will usually omit the reference to $\mathcal{P}$ or denote it by $\mathcal{P} \vdash e \to e'$ and $\mathcal{P} \vdash e \to^* e'$.

2.1 A proof calculus for constructor systems with extra variables

In [13] an adequate semantics for reachability of c-terms by term rewriting in CS's was presented. The key idea there was using a suitable notion of value, in this case the notion of s-cterm. *SCTerm* is the set of s-cterms, which are *finite* sets of elemental s-cterms, being the set *ESCTerm* of elemental s-cterms is defined as $ESCTerm \ni est ::= X \mid c(st_1,\ldots,st_n)$ for $X \in \mathcal{V}$, $c \in DC^n$, $st_1,\ldots,st_n \in SCTerm$. We extend this idea to expressions obtaining the sets *SExp* of s-expressions or just s-exp, and *ESExp* of elemental s-expressions, which are defined the same but now using any symbol in Σ in applications instead of just constructor symbols. Note that for s-expressions $\emptyset$ corresponds to $\bot$, so s-exps are partial by default. The approximation preorder $\sqsubseteq$ is defined for s-exps as the least preorder such that $se \sqsubseteq se'$ iff $\forall ese \in se.\exists ese' \in se'$ such that $ese \sqsubseteq ese'$, $X \sqsubseteq X$ for any $X \in \mathcal{V}$, and $h(se_1,\ldots,se_n) \sqsubseteq h(se'_1,\ldots,se'_n)$ iff $\forall i.se_i \sqsubseteq se'_i$. The sets *SSubst* and *SCSubst* of s-substitutions and s-csubstitutions (or just s-csubst) consist of finite mappings from variables to s-exps or s-cterms, respectively. We extend s-substs to be applied to *ESExp* and *SExp* as $\sigma : ESExp \to SExp$ defined by $X\sigma = \sigma(X)$, $h(\overline{se})\sigma = \{h(\overline{se\sigma})\}$; and $\sigma : SExp \to SExp$ defined by $se\sigma = \bigcup_{ese \in se} ese\sigma$. The approximation preorder $\sqsubseteq$ is defined for s-substs as $\sigma \sqsubseteq \theta$ iff $\forall X \in \mathcal{V}.\sigma(X) \sqsubseteq \theta(X)$. For any nonempty and finite set $\{\theta_1,\ldots,\theta_n\} \subseteq SCSubst$ we define $\bigcup\{\theta_1,\ldots,\theta_n\} \in SCSubst$ as $\bigcup\{\theta_1,\ldots,\theta_n\}(X) = \theta_1(X) \cup \ldots \cup \theta_n(X)$.

We obtain the denotation of an expression as the denotation of its associated s-expression, assigned by the operator $\widetilde{\ }: Exp_\bot \to SExp$, defined as $\widetilde{\bot} = \emptyset$; $\widetilde{X} = \{X\}$ for any $X \in \mathcal{V}$; $\widetilde{h(e_1,\ldots,e_n)} = \{h(\widetilde{e_1},\ldots,\widetilde{e_n})\}$ for any $h \in \Sigma^n$. The operator $\widetilde{\ }$ is extended to s-substitutions as $\widetilde{\sigma}(X) = \widetilde{\sigma(X)}$, for $\sigma \in Subst_\bot$. Is is easy to check that $\widetilde{e\sigma} = \widetilde{e}\widetilde{\sigma}$ (see [13]). Conversely, we can flatten an s-expression $se$to obtain the set $flat(se)$ of expressions "contained" in it, so $flat(\emptyset) = \{\bot\}$ and $flat(se) = \bigcup_{ese \in se} flat(ese)$ if $se \neq \emptyset$, where the flattening of elemental s-exps is defined as $flat(X) = \{X\}$; $flat(h(se_1,\ldots,se_n)) = \{h(e_1,\ldots,e_n) \mid e_i \in flat(se_i) \text{ for } i = 1..n\}$. In Figure 1 we can find the proof calculus that defines the semantics of s-expressions. Our proof calculus proves reduction statements of the form $se \twoheadrightarrow st$ with $se \in SExp$ and $st \in SCTerm$, expressing that st represents an approximation to one of the possible structured sets of values for se. We refer the interested reader to [13] for a detailed explanations about the calculus.We write $\mathcal{P} \vdash se \twoheadrightarrow st$ to express that $se \twoheadrightarrow st$ is derivable in our calculus under the CS $\mathcal{P}$. We say that a proof for a statement $\mathcal{P} \vdash se \twoheadrightarrow st$ is ground iff se, st and all the s-exp in the premises are ground. The *denotation* of an s-expression se under a CS $\mathcal{P}$ is defined as $[\![se]\!]^{\mathcal{P}} = \{st \in SCTerm \mid \mathcal{P} \vdash se \twoheadrightarrow st\}$, so $[\![e]\!]^{\mathcal{P}} = [\![\widetilde{e}]\!]^{\mathcal{P}}$. In the following we will usually omit the reference to $\mathcal{P}$. The denotation of $\sigma \in SSubst$ is

$\_ \vdash \_ \lessdot \_ \subseteq CS \times SCTerm \times Exp$	$\_ \vdash \_ \lessdot \_ \subseteq CS \times ESCTerm \times Exp$
$\mathcal{P} \vdash st \lessdot e$ if $\forall est \in st, \mathcal{P} \vdash est \lessdot e$	$\mathcal{P} \vdash X \lessdot e$ if $\mathcal{P} \vdash e \to^* X$
	$\mathcal{P} \vdash c(\overline{st}) \lessdot e$ if $\mathcal{P} \vdash e \to^* c(\overline{e})$ for some $\overline{e}$ such that $\forall e_i \in \overline{e}, \mathcal{P} \vdash st_i \lessdot e_i$

Figure 2: Domination relation

defined as $[\![\sigma]\!] = \{\theta \in SCSubst \mid \forall X \in \mathcal{V}, \sigma(X) \twoheadrightarrow \theta(X)\}$, so for $\theta \in Subst_\bot$ we define $[\![\theta]\!] = [\![\widetilde{\theta}]\!]$.

The setting presented in [13] was not able to deal with extra variables. As programs with extra variables are very common when using narrowing, for this work we decided to extend the setting to deal with them. But then we realized that the semantics was already prepared to deal with extra variables, as the rule **ROR** from Figure 1 allows to instantiate extra variables freely with s-cterms: therefore all that was left was proving the adecuacy of the semantics in this extended scenario. Nevertheless, as a consequence of the freely instantiation of extra variables in **ROR**, then every program with extra variables turns into non-deterministic. For example consider a program $\{f \to (X, X)\}$ for which the constructors $0, 1 \in CS^0$ are available, then we can do:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{0\} \to \{0\}}\,\mathbf{DC}}{\{0,1\} \twoheadrightarrow \{0\}}\,\mathbf{Less} \quad \{0,1\} \twoheadrightarrow \{1\}}{\widetilde{(X,X)}[X/\{0,1\}] = \{(\{0,1\},\{0,1\})\} \twoheadrightarrow \{(\{0\},\{1\})\}}\,\mathbf{DC}}{\widetilde{f} = \{f\} \twoheadrightarrow \{(\{0\},\{1\})\} = \widetilde{(0,1)}}\,\mathbf{ROR}$$

But in fact this is not very surprising, and it has to do with the relation between non-determinism and extra variables [1], but adapted to the run-time choice semantics [12, 17] induced by term rewriting. As a consequence of this we assume that all the programs contain the function ? defined by the rules $\mathcal{Q} = \{X \ ? \ Y \to X, X \ ? \ Y \to Y\}$, so we only consider non-confluent TRS's. We admit that this is a limitation of our setting, but we also conjecture that for confluent TRS's a simpler semantics could be used, for which the packing of alternatives of c-terms would not be needed. Anyway, the point is that having ? at one's disposal is enough to express the non-determinism of any program [10], so we can use it to define the transformation $\widehat{\_}$ from s-exp and elemental s-exp to partial expressions that, contrary to *flat*, now takes care of the keeping the nested set structure by means of uses of the ? function. Then $\widehat{\_} : ESExp \to Exp_\bot$ is defined by $\widehat{X} = X$, $\widehat{h(se_1, \ldots, se_n)} = h(\widehat{se_1}, \ldots, \widehat{se_n})$; and $\widehat{\_} : SExp \to Exp_\bot$ is defined by $\widehat{\emptyset} = \bot$, $\widehat{\{ese_1, \ldots, ese_n\}} = \widehat{ese_1} \ ? \ \ldots \ ? \ \widehat{ese_n}$ for $n > 0$, where in the case for $\widehat{\{ese_1, \ldots, ese_n\}}$ we use some fixed arbitrary order on terms in the line of Prolog [18] for arranging the arguments of ?. This operator is also overloaded for substitutions as $\widehat{\_} : SSubst \to Subst_\bot$ as $(\widehat{\sigma})(X) = \widehat{\sigma(X)}$. Thanks to the power of ? to express non-determinism, that transformation preserves the semantics from Figure 1, and we can use it to prove the following new result about the adequacy of the semantics for programs with extra variables—see the appendix for a detailed proof.

Theorem 1 (Adequacy of $[\![\_]\!]$) *For all $e, e' \in Exp, t \in CTerm_\bot, st \in SCTerm$:*
Soundness *$st \in [\![\widetilde{e}]\!]$ and $t \in flat(st)$ implies $e \to^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$. Therefore, $\widetilde{t} \in [\![\widetilde{e}]\!]$ implies $e \to^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$. Besides, in any of the previous cases, if t is total then $e \to^* t$.*
Completeness *$e \to^* e'$ implies $\widetilde{|e'|} \in [\![\widetilde{e}]\!]$. Hence, if t is total then $e \to^* t$ implies $\widetilde{t} \in [\![\widetilde{e}]\!]$.*

We refer the interested reader to [13] for more properties of $[\![\_]\!]$ like compositionality or monotonicity, some of which are used in the proofs for the results in the present paper. There is another characterization of $[\![\_]\!]$ closer to term rewriting which is based of the *domination relation* $\_ \lessdot \_$ presented in Figure 2 (we will omit the prefix "$\mathcal{P} \vdash$" when it is implied by the context). With this relation we try to transfer to the rewriting world the finer distinction between sets of values that the structured representation of *SCTerm* allows us to perform. We extend the relation $\_ \lessdot \_$ to $\_ \vdash \_ \lessdot \_ \subseteq CS \times SCSubst \times Subst$ by $\theta \lessdot \sigma$ iff $\forall X \in \mathcal{V}, \theta(X) \lessdot \sigma(X)$. As can be seen in [14], this relation is a key ingredient to prove the soundness of $[\![\_]\!]$, and its equivalence to $[\![\_]\!]$ is stated in the following result.

Lemma 1 (Domination) *For all $e \in Exp, st \in SCTerm$, $st \in [\![\widetilde{e}]\!]$ iff $st \lessdot e$. Besides, regarding substitutions, for all $\sigma \in Subst$, $\theta \in SCSubst$ we have that $\theta \in [\![\widetilde{\sigma}]\!]$ iff $\theta \lessdot \sigma$.*

3 The generators approach

In this section we will show a proposal for adapting the generators technique from the field of functional-logic programming [6, 1] to the lifting of term rewriting derivations from arbitrary instances of expressions. This technique consists in replacing free and extra variables by a call to a *generator function* that can be reduced to any ground c-term. The generator function *gen* is defined as follows:

Definition 1 (Generator function) *For any program $\mathcal{P}$ we can define a fresh function gen as follows: for each $c \in CS^n$ we add a new rule $gen \to c(gen, \ldots, gen)$ to the program. By $\mathcal{G}$ we denote the program that consists of the set of rules for gen.*

The point with *gen* is that we can use it to compute any *ground value*:

Proposition 1 *For all $t \in CTerm$, $st \in SCTerm$ and $\theta \in SCSubst$ such that those are ground we have $gen \to^* t$, $st \in [\![gen]\!]$ and $\theta \in [\![[\overline{X}/gen]]\!]$ for $\overline{X} = dom(\theta)$.*

Then the main idea with generators is that given some $e \in Exp$ with $var(e) = \overline{X}$, we can simulate narrowing with e by performing term rewriting with $e[\overline{X}/gen]$. As *gen* can be reduced to any ground s-cterm, then Lemma 1 from [13] suggests that this procedure will be able to lift derivations $e\sigma \to^* t$ with an arbitrary $\sigma \in Subst$, even those which are not normalized: e.g. we can easily apply this technique to the example in Section 1, getting $f(X, X)[X/gen] \to^* f(0, 1) \to 2$. Sadly, on the other hand, only derivations reaching a ground c-term will be lifted, and the reason for that is that *gen* can be reduced to an arbitrary ground c-term, but it cannot be reduced to any c-term with variables. Thus, under the program $\{g(c(X)) \to X\}$ the term rewriting derivation $g(Y)[Y/c(X)] \to X$ cannot be lifted by using generators, as $g(Y)[Y/gen] \to g(c(gen)) \to gen \not\to^* X$, even though $[Y/c(X)]$ is a normalized substitution.

In order to prove the completeness of the generators technique for the reachability of ground c-terms, we rely on the following modification of Lemma 1 from [13].

Lemma 2 *For all $\sigma \in SSubst$, $se \in SExp$, $st \in SCTerm$, if st is ground then $se\sigma \twoheadrightarrow st$ implies $\exists \theta \in [\![\sigma]\!]$ such that $se\theta \twoheadrightarrow st$, θ is ground and $dom(\theta) = dom(\sigma)$.*

Note the restriction to ground s-cterms in Lemma 2 is crucial, and that it reflects the lack of completeness for reaching non-ground c-terms of the generators technique: e.g. under the program $\{f \to c(X)\}$ using $se = \{Y\}$, $\sigma = [Y/\{f\}]$ and $st = \{c(\{X\})\}$ the only $\theta \in [\![[Y/\{f\}]]\!]$ fulfilling the first condition is $\theta = [Y/\{c(\{X\})\}]$, which is not ground. On the other hand those s-csubst obtained by Lemma 2 are ground, and so they are in the denotation of an appropriate substitution with only generators in its range.

Generators can be introduced in programs systematically in order to eliminate extra variables from program rules using a program transformation in the line of those from [6, 1]. In those works the usual call-time choice semantics for functional-logic programming [9] was adopted, therefore we use a different transformation that is adapted to the use of term rewriting, which leads to a different set of reachable c-terms than that obtained with call-time choice [17]. The point in eliminating extra variables is that this way we eliminate the "oracular guessing" that is performing in a term rewriting step using extra variables: by this guessing we refer for example to the instantiation performed under the program $\{f \to g(X), g(0) \to 1\}$ in the first step of the derivation $f \to g(0) \to 1$ for the extra variable X, that has to be instantiated with 0 in order for the derivation to continue. That, combined with a suitable on-demand evaluation strategy like natural rewriting [8], turns term rewriting with generators into an effective mechanism for lifting term rewriting derivations. We formalize our extra variable elimination transformation through the following definition.

Definition 2 (Generators program transformation)
Given a program $\mathcal{P}$ its transformation $\hat{\mathcal{P}}$ consists of the rules $\mathcal{G}$ for gen together with the transformation of each rule in $\mathcal{P}$, defined as $\widehat{f(p_1, \ldots, p_n) \to r} = f(p_1, \ldots, p_n) \to r[\overline{X}/gen]$, where $\overline{X} = vExtra(f(p_1, \ldots, p_n) \to r)$.

Then it is clear that for any program $\mathcal{P}$ its transformation $\hat{\mathcal{P}}$ does not have any extra variable in its rules. Note that, contrary to the proposals from [6, 1], this transformation destroys the sharing that normally appears when there are several occurrences of the same variable, in procedures that instantiate variables like narrowing or SLD resolution. In our transformation, however, once instantiated with *gen* every occurrence of the same variable evolves independently. This is needed to ensure completeness under the transformed program, which can be seen considering the program$\mathcal{P} = \{f \to (g(X), h(X), g(0) \to$

$1, h(1) \to 2\}$ and the derivation $\mathcal{P} \vdash f \to (g(0\ ?\ 1), h(0\ ?\ 1)) \to^* (g(0), h(1)) \to^* (1, 2)$: as extra variables can be instantiated with arbitrary expressions that implies that in particular those can be instantiated with "alternatives" of expressions built using the ? function, which can evolve independently after the alternative between them is resolved. We can lift that derivation with our transformation as $\hat{\mathcal{P}} \vdash f \to (g(gen), h(gen)) \to^* (g(0), h(1)) \to^* (1, 2)$. The adequacy of the transformation is formulated in the following result, in the same terms as the variable elimination result from [6].

Theorem 2 *For any program $\mathcal{P}$, $se \in SExp$, $st \in SCTerm$ if st is ground then $\mathcal{G} \uplus \mathcal{P} \vdash se \twoheadrightarrow st$ iff $\hat{\mathcal{P}} \vdash se \twoheadrightarrow st$.*

After eliminating extra variableswith the proposed program transformation, we can then emulatethe instantiation of variables performed by a narrowing procedure by just replacing free variables with *gen*, thus lifting any term rewriting derivation starting from an arbitrary instance of an expression to a ground c-term.

Theorem 3 (Lifting) *For any program $\mathcal{P}$, $e, e' \in Exp$ such that e' is ground:*
Soundness *$\hat{\mathcal{P}} \vdash e[\overline{X/gen}] \to^* e'$ implies $\exists \sigma \in Subst$ such that $\mathcal{P} \vdash e\sigma \to^* e''$ for some $e'' \in Exp$ such that $|e'| \sqsubseteq |e''|$ with $dom(\sigma) = \overline{X}$. As a consequence, if $e' = t \in CTerm$ then $\mathcal{P} \vdash e\sigma \to^* t$.*
Completeness *For any $\sigma \in Subst$ we have that $\mathcal{P} \vdash e\sigma \to^* e'$ implies $\hat{\mathcal{P}} \vdash e[\overline{X/gen}] \to^* e''$ for some $e'' \in Exp$ such that $|e'| \sqsubseteq |e''|$ with $\overline{X} = dom(\sigma)$. As a consequence, if $e' = t \in CTerm$ then $\hat{\mathcal{P}} \vdash e[\overline{X/gen}] \to^* t$.*

4 Maude prototype

We present in this section our prototype; much more information can be found at `http://gpd.sip.ucm.es/snarrowing`. The prototype is started by typing `loop init-s .`, that initiates an input/output loop where programs and commands can be introduced. These programs have syntax `smod NAME is STMNTS ends`, where `NAME` is the identifier of the program and `STMNTS` is a sequence of constructor-based left-linear rewrite rules, written as follows:

```
(smod IPL is
  f(c(X),Y) -> h(X,Y) .
  h(X, c(Y)) -> g(X,Y) .
  g(0,1) -> 2 .
ends)
```

where upper-case letters are assumed to be variables. We can evaluate terms with variables with the command `eval-gen`, that transforms each variable in the term into the `gen` constant described above and evaluates the thus obtained expression in the module extended with the `gen` rules:

```
Maude> (eval-gen f(X,X) .)
Result: 2
```

That is, the tool informs us that it is possible to start with a term with the same variable and reach the value 2. We can ask the system for more solutions with the `cont` command until no more solutions (as in the current example) are found. The system combines the on-demand strategy with two different search strategies: depth-first and breadth-first. The user can switch between them with the commands :

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

5 Concluding remarks and ongoing work

In this work we have proposed and formally proved the adequacy of a technique for lifting term rewriting derivations from an arbitrary instance of an expression to a constructed term—or the outer construted part of any expression—using constructor systems. It is based on the generator technique from the field of functional-logic programming [6, 1], but adapted to the different semantic context of term rewriting [17]. A fundamental limitation of the generators is that it can only be used for reaching ground c-terms or the outer constructed part of expressions).This limitation can be somewhat mitigated by reducing the reachability to a non-ground value to the reachability of a ground value: for example to test for $e \to^* c(X)$ we can define a new function f by the rule $f(c(X)) \to true$ and then test for $f(e) \to^* true$. Anyway this is a partial solution, and moreover the instantiation of free variables corresponding to the evaluation of *gen* cannot be obtained by a transformation in that line, for example by evaluating $(f(X), X)$ in the previous

example, due to the aforementioned loss of sharing between different occurrences of the same variable. This later limitation could only be possibly overcomed by using some metaprogramming capabilities of the rewriting engine used to implement this technique. The generators technique has been used in practical systems, for example as the basis for an implementation of the functional-programming language Curry [4]. There the information provided by a Damas-Milner like type system is used to improve the efficiency, because instead of just one universal generator, like in our proposal, several generators are used, one for each type, which results in a great shrink of the search space for the evaluation of generators. One could argue that our generator are fundamentally equivalent to defining a generator $genE$ that could be reduced to any expression, and then replacing each free or extra variable with $genE$, which would be trivially complete. Nevertheless, in our approach the search space for generators is significantly smaller, especially when combined with type information.

The system has been implemented in a Maude prototype that allows us to study their expressivity and possible applications. This prototype uses the on-demand strategy natural rewriting [8], thus providing an efficient implementation. Finally, along the way we have extended the semantics for constructor systems from [13] to support extra variables in rewriting rules, as those are very frequent when using narrowing.

Regarding future work, we plan to improve our implementation by using the reflection capabilities of Maude to collect the evaluation of generators, in order to be able to present a computed answer for generators derivations.

References

[1] Sergio Antoy and Michael Hanus. Overlapping rules and logic variables in functional logic programs. In Sandro Etalle and Miroslaw Truszczynski, editors, *ICLP*, volume 4079 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2006.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[3] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[4] Bernd Braßel, Michael Hanus, Björn Peemöller, and Fabian Reck. KiCS2: A new compiler from Curry to Haskell. In Herbert Kuchen, editor, *Proceedings of the 20th international conference on Functional and constraint Logic Programming, WFLP 2011*, volume 6816 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude: A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[6] J. Dios-Castro and F. López-Fraguas. Extra variables can be eliminated from functional logic programs. *Electronic Notes in Theoretical Computer Science 188*, pages 3–19, 2007.

[7] Francisco Durán, Steven Eker, Santiago Escobar, José Meseguer, and Carolyn L. Talcott. Variants, unification, narrowing, and symbolic reachability in maude 2.6. In Manfred Schmidt-Schauß, editor, *RTA*, volume 10 of *LIPIcs*, pages 31–40. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[8] Santiago 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.

[9] 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.

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

[11] J.M. Hullot. Canonical forms and unification. In *Proc. 5th Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer, 1980.

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

[13] Francisco López-Fraguas, Juan Rodríguez-Hortalá, and Jaime 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.

[14] Francisco López-Fraguas, Juan Rodríguez-Hortalá, and Jaime Sánchez-Hernández. A Fully Abstract Semantics for Constructor Systems (Extended version). Technical Report SIC-2-09, Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 2009.

[15] José Meseguer and Prasanna Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.

[16] Aart Middeldorp and Erik Hamoen. Completeness results for basic narrowing. *Appl. Algebra Eng. Commun. Comput.*, 5:213–253, 1994.

[17] J. Rodríguez-Hortalá. A hierarchy of semantics for non-deterministic term rewriting systems. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *Proceedings Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008*, volume 2 of *Leibniz International Proceedings in Informatics*, pages 328–339. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008.

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

A Proofs for the results

During proofs we will often use IH as an abbreviation for "induction hypothesis."

A.1 Proofs for Section 2

The following useful properties indicate that the definition of $\widehat{\ }$ corresponds to its intended meaning.

Lemma 3

a) For all $\sigma \in SSubst$ we have $dom(\sigma) = dom(\widehat{\sigma})$.

b) For all $st \in SCTerm, est \in ESCTerm$ we have that $st \lessdot \widehat{st}$ and $est \lessdot \widehat{est}$.

c) For all $\theta \in SCSubst$ we have that $\theta \lessdot \widehat{\theta}$.

Proof.

a) For the left to right inclusion, assume some $X \notin dom(\widehat{\sigma})$. Then $(\widehat{\sigma})(X) = X$ and so $\widehat{\sigma(X)} = (\widehat{\sigma})(X) = X$, which implies $\sigma(X) = \{X\}$, as $\{X\}$ is the only set-expression such that applying $\widehat{\ }$ to it returns X—it cannot be the empty set as $X \neq \bot$, it must be singleton to avoid ? and therefore it only element must be X. But then $X \notin dom(\sigma)$.

For the converse inclusion assume some $X \in dom(\widehat{\sigma})$. Then $(\widehat{\sigma})(X) \neq X$ and so $\widehat{\sigma(X)} = (\widehat{\sigma})(X) \neq X$, which implies $\sigma(X) \neq \{X\}$, because otherwise we would reach a contradiction as then $(\widehat{\sigma})(X) = \widehat{\sigma(X)} = X$. But then $X \in dom(\sigma)$.

b) We proceed by induction on the structure of *SCTerm* and *ESCTerm*. Concerning the base cases:

- $st = \emptyset$: Then $st = \emptyset \lessdot \widehat{st}$ by definition.
- $est = X \in \mathcal{V}$: Then $\widehat{est} = X$, and so we have $est = X \lessdot X = \widehat{est}$, as $X \rightarrow^0 X$.
- $est = c \in CS^0$: Then $\widehat{est} = c$, and so we have $est = c \lessdot c = \widehat{est}$, as $c \rightarrow^0 c$.

Regarding the inductive steps:

- $st = \{est_1, \ldots, est_n\}$ with $n > 0$: Then $\widehat{st} = \widehat{est_1} \ ? \ \ldots \ ? \ \widehat{est_n}$, and we can apply the IH to get that $\forall i.est_i \lessdot \widehat{est_i}$. Therefore, as $\forall i.\widehat{st} = \widehat{est_1} \ ? \ \ldots \ ? \ \widehat{est_n} \rightarrow^* \widehat{est_i}$ by several applications of the rules for ?, we conclude that $\forall i.est_i \lessdot \widehat{st}$ by Lemma 19 from [14] . But then $\{est_1, \ldots, est_n\} = st \lessdot \widehat{st} = \widehat{est_1} \ ? \ \ldots \ ? \ \widehat{est_n}$.
- $est = c(st_1, \ldots, st_n)$: Then $\widehat{est} = c(\widehat{st_1}, \ldots, \widehat{st_n})$, and we can apply the IH to get that $\forall i.st_i \lessdot \widehat{st_i}$. Therefore, as $\widehat{est} = c(\widehat{st_1}, \ldots, \widehat{st_n}) \rightarrow^0 c(\widehat{st_1}, \ldots, \widehat{st_n})$ we conclude that $c(st_1, \ldots, st_n) = est \lessdot \widehat{est} = c(\widehat{st_1}, \ldots, \widehat{st_n})$.

c) As $dom(\theta) = dom(\widehat{\theta})$ by Lemma 3, then by Lemma 17 from [14] we only have to prove that $\forall X \in dom(\theta).\theta(X) \lessdot (\widehat{\theta})(X)$, which is a consequence of part *b)* and the definition of $\widehat{\theta}$, so we are done.

□

In order to extend the results from [13] to deal with programs with extra variables we just have to modify the proof of the following lemma—named Lemma 2 in [14]—, on which it relies the proof for the soundness of $[\![\_]\!]$ and its relation to $\_ \lessdot \_$.

Lemma 4 *Let $e \in Exp, st \in SCTerm, \theta \in SCSubst$.*

$$\textit{If } \widetilde{e}\theta \twoheadrightarrow st \textit{ then } st \lessdot e\sigma \textit{ for any } \sigma \in Subst \textit{ such that } \theta \lessdot \sigma$$

Proof. As expected the only change w.r.t. the proof from [14] is the case for **ROR** in an induction on the structure of $\widetilde{e}\theta \twoheadrightarrow st$, because otherwise the calculus behaves the same for programs with extra variables and programs without extra variales. Then we have $e = f(e_1, \ldots, e_n)$, let us assume some $\theta \in SCSubst$ such that $\widetilde{e}\theta \twoheadrightarrow st$ with a proof of the following shape:

$$\frac{\widetilde{e_1}\theta \twoheadrightarrow \widetilde{p_1}\mu \quad \ldots \quad \widetilde{e_n}\theta \twoheadrightarrow \widetilde{p_n}\mu \quad \widetilde{r}\mu \twoheadrightarrow st}{\widetilde{e}\theta \equiv \{f(\widetilde{e_1}\theta, \ldots, \widetilde{e_n}\theta)\} \twoheadrightarrow st} \ \textbf{ROR}$$

for some $(f(p_1, \ldots, p_n) \rightarrow r) \in \mathcal{P}$. Then given $\mathcal{V}_e = vExtra(f(p_1, \ldots, p_n) \rightarrow r)$ we may assume that $dom(\mu) \subseteq var(f(p_1, \ldots, p_n)) \uplus \mathcal{V}_e$ without loss of generality. Besides, as $f(p_1, \ldots, p_n)$ is linear, if for each $i \in \{1, \ldots, n\}$ we define $\mu_i = \mu|_{var(\widetilde{p_i})} = \mu_{var(p_i)}$ (as $var(\widetilde{p_i}) = var(p_i)$ by Proposition 8 from [14]), and $\mu_e = \mu|_{\mathcal{V}_e}$, then $\mu = \mu_1 \uplus \ldots \uplus \mu_n \uplus \mu_e$ and $\forall i, \widetilde{p_i}\mu \equiv \widetilde{p_i}\mu_i$.
Now given some $\sigma \in Subst$ such that $\theta \lessdot \sigma$ we can apply the IH to each $\widetilde{e_i}\theta \twoheadrightarrow \widetilde{p_i}\mu$ to get $\forall i, \widetilde{p_i}\mu_i \equiv \widetilde{p_i}\mu \lessdot e_i\sigma$. But then by Lemma 18 from [14] we have that $\forall i, \exists \sigma'_i \in Subst$ such that $dom(\sigma'_i) = dom(\mu_i)$, $\mu_i \lessdot \sigma'_i$ and $e_i\sigma \rightarrow^* p_i\sigma'_i$. As $\forall i, dom(\sigma'_i) = dom(\mu_i) \subseteq var(p_i)$, this combined with the linearity of $f(p_1, \ldots, p_n)$, the definition of μ_e and Lemma 3 implies that $\sigma' = \sigma'_1 \uplus \ldots \uplus \sigma'_n \uplus \widehat{\mu_e}$ is correctly defined. We will now see that $\mu \lessdot \sigma'$ by applying Lemma 17 from [14], because $dom(\sigma') = \bigcup_i dom(\sigma'_i) \uplus dom(\widehat{\mu_e}) = \bigcup_i dom(\mu_i) \uplus dom(\mu_e) = dom(\mu)$, so we only have to check that given some $X \in dom(\mu)$ we have $\mu(X) \lessdot \sigma'(X)$, which is true because:

- If $X \in dom(\mu_i)$ for some i and then $\mu(X) \equiv \mu_i(X) \lessdot \sigma'_i(X) \equiv \sigma'(X)$, as $\mu_i \lessdot \sigma'_i$.
- Otherwise $X \in dom(\mu_e)$ and then $\mu(X) = \mu_e(X) \lessdot (\widehat{\mu_e})(X) = \sigma'(X)$, as $\mu_e \lessdot \widehat{\mu_e}$ by Lemma 3.

As $\mu \lessdot \sigma'$ we can finally use σ' to apply the IH to $\widetilde{r}\mu \twoheadrightarrow st$, getting that $st \lessdot r\sigma'$. Therefore by Lemma 19 from [14] we just need to prove $e\sigma \rightarrow^* r\sigma'$ to reach to the final conclusion $st \lessdot e\sigma$, but then we can do $e\sigma \equiv f(e_1\sigma, \ldots, e_n\sigma) \rightarrow^* f(p_1\sigma'_1, \ldots, p_n\sigma'_n) \equiv f(p_1, \ldots, p_n)\sigma' \rightarrow r\sigma'$, where the variables in $\mathcal{V}_e$ are instantiated by $\widehat{\mu_e}$, and we are done. □

Proof. [For Lemma 1 (page 5)] For the first part see [14], now using the new proof for Lemma 4. Regarding the second part, by definition $[\![\widetilde{\sigma}]\!] = \{\theta \in SCSubst \mid \forall X \in \mathcal{V}, \widetilde{\sigma}(X) \twoheadrightarrow \theta(X)\}$. But by definition of $\widetilde{\ }$ we have that $\widetilde{\sigma}(X) = \widetilde{\sigma(X)}$, therefore $\widetilde{\sigma}(X) \twoheadrightarrow \theta(X)$ iff $\widetilde{\sigma(X)} \twoheadrightarrow \theta(X)$ iff $\theta(X) \lessdot \sigma(X)$ by the first part. Hence we are done as by definition $\theta \lessdot \sigma$ iff $\forall X \in \mathcal{V}.\theta(X) \lessdot \sigma(X)$. □

Based on $[\![\_]\!]$ be define the preorder $\trianglelefteq$ over $SSubst$ by $\sigma \trianglelefteq \sigma'$ iff $\forall X \in \mathcal{V}.[\![\sigma(X)]\!] \subseteq [\![\sigma'(X)]\!]$. By $\lhd|\rhd$ we denote the equivalence relation generated by $\trianglelefteq$, i.e., $\sigma_1 \lhd | \rhd \sigma_2$ iff $\sigma_1 \trianglelefteq \sigma_2$ and $\sigma_2 \trianglelefteq \sigma_1$. It is trivial to check that $\sigma_1 \lhd | \rhd \sigma_2$ iff $\forall X \in \mathcal{V}.[\![\sigma_1(X)]\!] = [\![\sigma_2(X)]\!]$. We can use $\lhd|\rhd$ to express the relation between $\widetilde{\ }$ and $\widehat{\ }$:

Lemma 5

- *Forall $se \in SExp$, $ese \in ESExp$ we have that $[\![\widetilde{\widehat{se}}]\!] = [\![se]\!]$ and $[\![\widetilde{\widehat{ese}}]\!] = [\![\{ese\}]\!]$.*
- *For all $\sigma \in SSubst$ we have that $\widetilde{\widehat{\sigma}} \lhd \mid \rhd \sigma$.*
- *For all $e \in Exp_\perp$ and $\sigma \in Subst_\perp$ we have that $\widehat{\widetilde{e}} = e$ and $\widehat{\widetilde{\sigma}} = \sigma$.*

Proof. The first part can be proved by a routine induction on the structure of se and ese. Regarding the second part we just need to prove that $\forall X \in \mathcal{V}.[\![(\widetilde{\widehat{\sigma}})(X)]\!] = [\![\sigma(X)]\!]$, but $[\![(\widetilde{\widehat{\sigma}})(X)]\!] = [\![\widetilde{\widehat{\sigma(X)}}]\!] = [\![\sigma(X)]\!]$ by the first part, so we are done. For the third part, we can prove that $\widehat{\widetilde{e}} = e$ by a simple induction on the structure of expressions, based on the fact that $\widetilde{\_}$ only introduces singleton sets. But then $\widehat{\widetilde{\sigma}} = \sigma$ follows easily from $\widehat{\widetilde{e}} = e$ for any expression e. □

Lemma 6 *For all $\sigma \in SSubst$, $e \in Exp$, $t \in CTerm_\perp$ we have that $\widetilde{e}\sigma \twoheadrightarrow \widetilde{t}$ implies $e\widehat{\sigma} \rightarrow^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$. As a consequence, if t is total then $e\widehat{\sigma} \rightarrow^* t$.*

Proof. Assume $\widetilde{e}\sigma \twoheadrightarrow \widetilde{t}$, then by Lemma 5 and the monotonicity of $[\![\_]\!]$ (see [13]) we get $\widetilde{e}\widetilde{\widehat{\sigma}} \twoheadrightarrow \widetilde{t}$, as $\widetilde{\widehat{\sigma}} \lhd \mid \rhd \sigma$. But $\widetilde{e}\widetilde{\widehat{\sigma}} = \widetilde{e\widehat{\sigma}}$, therefore $\widetilde{e\widehat{\sigma}} \twoheadrightarrow \widetilde{t}$, hence $e\widehat{\sigma} \rightarrow^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$ by correctness of $[\![\_]\!]$. Besides, if t is total then it is trivial to check that $t \sqsubseteq |e'|$ implies $t = |e'|$ thus getting $e' = t$. □

A.2 Proofs for Section 3

Proof. [For Proposition 1] The first and second parts follow by simple induction on the structure of t and st (which implies another related induction on the structure of est for $est \lessdot gen$). The third part follows from the second one combined with Lemma 17 from [14]. □

Proof. [For Lemma 2] We just have to perform a slight modification of the proof for Lemma 1 from [14], now checking the additional conditions of θ is ground and $dom(\theta) = dom(\sigma)$, using that st is ground.

- If $se \equiv \{X\}$ for some $X \in dom(\sigma)$: Then the hypothesis is $\sigma(X) \twoheadrightarrow st$, and we can define $\theta \in SCSubst$ as
$$\theta(Y) = \begin{cases} st & \text{if } Y \equiv X \\ \emptyset & \text{if } Y \in dom(\sigma) \setminus \{X\} \\ \{Y\} & \text{otherwise} \end{cases}$$
As st is ground by hypothesis then $\theta(X) = st \neq \{X\}$, therefore $X \in dom(\theta)$, and so $dom(\theta) = dom(\sigma)$ by definition of θ. Besides, given $Y \in dom(\theta) = dom(\sigma)$ either $Y = X$ and so $\theta(Y) = st$ is ground by hypothesis; or $\theta(Y) = \emptyset$, which is ground.
- In several other cases we take $\theta = [\overline{Y/\emptyset}]$ with $\overline{Y} = dom(\sigma)$, for which is trivial to check that it is ground and has the same domain as σ.
- Finally, in other cases some s-csubst $\{\theta_1, \ldots, \theta_n\}$ obtained by IH thus being ground and which the same domain as σ, are combined into $\theta = \bigcup\{\theta_1, \ldots, \theta_n\}$. But then $dom(\theta) = \bigcup_i dom(\theta_i) = \bigcup_i dom(\sigma) = dom(\sigma)$ by definition of $\bigcup\{\theta_1, \ldots, \theta_n\}$ and IH. Besides θ is ground because each θ_i is ground by IH.

□

The proof of Theorem 2 follows the same structure of the proof for the extra variable elimination transformation from [6]. In this proof we rely on some auxiliary results and notions.

Lemma 7 *For any $se \in SExp$, $st \in SCTerm$ if se and st are ground and $se \twoheadrightarrow st$ then there is ground proof for that statement*

Proof. A straightforward modification of the corresponding result—Lemma 3.7—from [6]. □

Now we are ready to prove the adequacy of the generator transformation.

Proof. [For Theorem 2] For the left to right implication let us assume $\mathcal{G} \uplus \mathcal{P} \vdash se \twoheadrightarrow st$. First we will prove the case when se is ground, then by Lemma 7 we may assume that the proof for that statement is ground without loss of generality. Now we proceed by induction on the structure of the proof. In this induction all the cases are trivial except for the one for an application of **ROR** using a rule from $\mathcal{P}$, because the rest of the rules of the calculus are independent from the program, and if a rule from $\mathcal{G}$ is used we can use the same rule as it also belongs to $\hat{\mathcal{P}}$. Let us assume then a **ROR** step using some rule $(f(p_1, \ldots, p_n) \to r) \in \mathcal{P}$, so we have:

$$\frac{\mathcal{G} \uplus \mathcal{P} \vdash se_1 \twoheadrightarrow \widetilde{p_1}\theta \;\; \ldots \;\; se_n \twoheadrightarrow \widetilde{p_n}\theta \;\; \widetilde{r}\theta \twoheadrightarrow st}{\mathcal{G} \uplus \mathcal{P} \vdash \{f(se_1, \ldots, se_n)\} \twoheadrightarrow st}\ \mathbf{ROR}$$

with $(f(p_1, \ldots, p_n) \to r[\overline{X/gen}]) \in \hat{\mathcal{P}}$ with $\overline{X} = vExtra(f(p_1, \ldots, p_n) \to r)$. Then we can apply the IH over $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{r}\theta \twoheadrightarrow st$ to obtain $\mathcal{P} \vdash \widetilde{r}\theta \twoheadrightarrow st$. As the starting proof was ground then $\widetilde{r}\theta$ is ground and so we may assume that θ is ground without loss of generality. Now if we define $\theta_e = \theta|_{\overline{X}}$ and $\theta_p = \theta|_{\setminus \overline{X}}$ then we have $\theta = \theta_e \uplus \theta_p$ with θ_e ground. But then $\theta_e \lessdot [\overline{X/gen}]$ by Proposition 1 thus $\theta_e \in [\![\widetilde{[\overline{X/gen}]}]\!]$ by Lemma 1 and so $\theta_e \trianglelefteq \widetilde{[\overline{X/gen}]}$ by Lemma 12 from [14]. But then it is easy to check that $\theta = \theta_e \uplus \theta_p \sqsubseteq \theta_p \uplus \widetilde{[\overline{X/gen}]}$, hence $\hat{\mathcal{P}} \vdash \widetilde{r}\theta \twoheadrightarrow st$ implies $\hat{\mathcal{P}} \vdash \widetilde{r}(\theta_p \uplus \widetilde{[\overline{X/gen}]}) \twoheadrightarrow st$ by the monotonicity of substitutions of $[\![\_]\!]$. Now if we use the characterization $\sigma\theta = [X\sigma\theta \mid X \in dom(\sigma)] \uplus \theta|_{\setminus dom(\sigma)}$ from [3], which also holds for $SSubst$, we can conclude $\theta_p \uplus \widetilde{[\overline{X/gen}]} = \widetilde{[\overline{X/gen}]}\theta_p$, therefore $\hat{\mathcal{P}} \vdash \widetilde{r[\overline{X/gen}]}\theta_p = \widetilde{r}\widetilde{[\overline{X/gen}]}\theta_p \twoheadrightarrow st$. Finally without loss of generality we may assume $dom(\theta) \subseteq var(f(\overline{p}) \to r) = var(f(\overline{p})) \uplus \overline{X}$, therefore $dom(\theta_p) = var(f(\overline{p}))$ and so $\forall i. \hat{\mathcal{P}} \vdash se_i \twoheadrightarrow \widetilde{p_i}\theta = \widetilde{p_i}\theta_p$ follows by IH over the corresponding premises of the starting proof, and we can prove $\hat{\mathcal{P}} \vdash \{f(se_1, \ldots, se_n)\} \twoheadrightarrow st$ by **ROR** using the program rule $(f(p_1, \ldots, p_n) \to r[\overline{X/gen}]) \in \hat{\mathcal{P}}$.

Now for the general case where se is not required to be ground, given $\overline{Y} = var(se)$ we have that $\mathcal{G} \uplus \mathcal{P} \vdash se[\overline{Y/\emptyset}] \twoheadrightarrow st[\overline{Y/\emptyset}] = st$ by closedness under substitutions of $[\![\_]\!]$—see [13]—and because st is ground. Then by the previous case we get $\hat{\mathcal{P}} \vdash se[\overline{Y/\emptyset}] \twoheadrightarrow st$. But $se[\overline{Y/\emptyset}] \sqsubseteq se$, therefore we get $\hat{\mathcal{P}} \vdash se \twoheadrightarrow st$ the polarity of $[\![\_]\!]$.

Regarding the converse implication we will also prove the case for se ground, so the general case where se is not restricted follows using the same reasoning we performed for the other implication. Similarly, we assume $\hat{\mathcal{P}} \vdash se \twoheadrightarrow st$, we apply Lemma 7 to assume that the proof for that statement is ground, and proceed by induction on the structure of the proof. All the cases are trivial except the one for an application of **ROR** using a rule $(f(p_1, \ldots, p_n) \to r[\overline{X/gen}]) \in \hat{\mathcal{P}}$ corresponding to a rule $(f(p_1, \ldots, p_n) \to r) \in \mathcal{P}$, so we have:

$$\frac{\hat{\mathcal{P}} \vdash se_1 \twoheadrightarrow \widetilde{p_1}\theta \;\; \ldots \;\; se_n \twoheadrightarrow \widetilde{p_n}\theta \;\; \widetilde{r[\overline{X/gen}]}\theta = \widetilde{r}\widetilde{[\overline{X/gen}]}\theta \twoheadrightarrow st}{\hat{\mathcal{P}} \vdash \{f(se_1, \ldots, se_n)\} \twoheadrightarrow st}\ \mathbf{ROR}$$

Then we can apply the IH over $\hat{\mathcal{P}} \vdash \widetilde{r}\widetilde{[\overline{X/gen}]}\theta \twoheadrightarrow st$ to obtain $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{r}\widetilde{[\overline{X/gen}]}\theta \twoheadrightarrow st$. As the starting proof was ground then we may assume that θ is ground without loss of generality, so we can apply the characterization of composition of substitutions from [3] to obtain $\widetilde{[\overline{X/gen}]}\theta = \widetilde{[\overline{X/gen}]} \uplus \theta_p = \theta_p\widetilde{[\overline{X/gen}]}$ for $\theta_p = \theta|_{\setminus \overline{X}}$. Therefore $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{r}\widetilde{[\overline{X/gen}]}\theta = (\widetilde{r}\theta_p)\widetilde{[\overline{X/gen}]} \twoheadrightarrow st$ which combined with Lemma 2 implies that $\exists \mu \in [\![\widetilde{[\overline{X/gen}]}]\!] \subseteq SCSubst$ such that $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{r}\theta_p\mu \twoheadrightarrow st$, μ is ground and $dom(\mu) = dom([\overline{X/gen}]) = \overline{X}$. But as θ_p is ground and $\overline{X} \cap dom(\theta_p) = \emptyset$ by definition then $\theta_p\mu = \theta_p \uplus \mu$, therefore $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{r}(\theta_p \uplus \mu) \twoheadrightarrow st$. Finally without loss of generality we may assume $dom(\theta) \subseteq var(f(\overline{p}) \to r[\overline{X/gen}]) = var(f(\overline{p})) \uplus \overline{X}$, by the definition of $\hat{\mathcal{P}}$, therefore $dom(\theta_p) = var(f(\overline{p}))$ and so so $\forall i. \mathcal{G} \uplus \mathcal{P} \vdash se_i \twoheadrightarrow \widetilde{p_i}\theta = \widetilde{p_i}\theta_p = \widetilde{p_i}(\theta_p \uplus \mu)$ follows by IH over the corresponding premises of the starting proof, and we can prove $\hat{\mathcal{P}} \vdash \{f(se_1, \ldots, se_n)\} \twoheadrightarrow st$ by **ROR** using the program rule $(f(p_1, \ldots, p_n) \to r) \in \mathcal{G} \uplus \mathcal{P}$. □

Proof. [For Theorem 3]

Soundness Assume $\hat{\mathcal{P}} \vdash e[\overline{X/gen}] \rightarrow^* e'$ then $\hat{\mathcal{P}} \vdash \widetilde{e[\overline{X/gen}]} = \widetilde{e}[\widetilde{\overline{X/gen}}] \twoheadrightarrow \widetilde{|e'|}$, by the completeness of $[\![\_]\!]$ and the properties of $\widetilde{\_}$. As e' is ground then $\widetilde{|e'|}$ is also ground and so $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{e}[\widetilde{\overline{X/gen}}] \twoheadrightarrow \widetilde{|e'|}$ by Theorem 2. Besides, by Lemma 2 $\exists \theta \in [\![\widetilde{[\overline{X/gen}]}]\!] \subseteq \mathit{SCSubst}$ such that $\mathcal{G} \uplus \mathcal{P} \vdash \widetilde{e}\theta \twoheadrightarrow \widetilde{|e'|}$, θ is ground and $\mathit{dom}(\theta) = \mathit{dom}(\widetilde{[\overline{X/gen}]}) = \overline{X}$. As gen does not appear in θ as it is an s-csubst, nor in e or $\mathcal{P}$ as gen is defined as a fresh function in the transformation, then the rules in $\mathcal{G}$ are not used in that proof and so $\mathcal{P} \vdash \widetilde{e}\theta \twoheadrightarrow \widetilde{|e'|}$. Now we can build a substitution $\theta \in \mathit{SCSubst}$ such that $\theta \sqsubseteq \theta'$ and $\emptyset$ does not appear in the range of θ', by replacing each occurrence of $\emptyset$ in the range of θ by a fresh variable, for example. But then $\mathcal{P} \vdash \widetilde{e}\theta' \twoheadrightarrow \widetilde{|e'|}$ by the monotonicity of $[\![\_]\!]$, and we can apply Lemma 6 to get $\mathcal{P} \vdash e\widehat{\theta'} \rightarrow^* e''$ for some e'' such that $|e'| \sqsubseteq |e''|$. Finally it is easy to see that as θ' contains no occurrence of $\emptyset$ then $\widehat{\theta'} \in \mathit{Subst}$. To conclude, if $e' = t \in \mathit{CTerm}$ is total then it is trivial to check that $|t| \sqsubseteq |e''|$ implies $t = |e''|$ thus getting $e'' = t$.

Completeness Assume $\mathcal{P} \vdash e\sigma \rightarrow^* e'$, then $\mathcal{P} \vdash \widetilde{e\sigma} = \widetilde{e}\widetilde{\sigma} \twoheadrightarrow \widetilde{|e'|}$, by the completeness of $[\![\_]\!]$ and the properties of $\widetilde{\_}$. Therefore $\mathcal{P} \uplus \mathcal{G} \vdash \widetilde{e}\widetilde{\sigma} \twoheadrightarrow \widetilde{|e'|}$, and as e' is ground then $\widetilde{|e'|}$ is also ground and so $\hat{\mathcal{P}} \vdash \widetilde{e}\widetilde{\sigma} \twoheadrightarrow \widetilde{|e'|}$ by Theorem 2. Besides, by Lemma 2 $\exists \theta \in [\![\widetilde{\sigma}]\!]$ such that $\hat{\mathcal{P}} \vdash \widetilde{e}\theta \twoheadrightarrow \widetilde{|e'|}$, θ is ground and $\mathit{dom}(\theta) = \mathit{dom}(\widetilde{\sigma}) = \overline{X}$. But then $\theta \lessdot [\overline{X/gen}]$, by Proposition 1, which implies $\theta \in [\![\widetilde{[\overline{X/gen}]}]\!]$ by Lemma 1, which implies $\theta \trianglelefteq \widetilde{[\overline{X/gen}]}$ by Lemma 12 from [14]. We can combine this with $\hat{\mathcal{P}} \vdash \widetilde{e}\theta \twoheadrightarrow \widetilde{|e'|}$ and the monotonicity of $[\![\_]\!]$ to conclude that $\hat{\mathcal{P}} \vdash \widetilde{e[\overline{X/gen}]} = \widetilde{e}[\widetilde{\overline{X/gen}}] \twoheadrightarrow \widetilde{|e'|}$, hence $\hat{\mathcal{P}} \vdash e[\overline{X/gen}] \rightarrow^* e''$ for some $e'' \in \mathit{Exp}$ such that $|e'| \sqsubseteq |e''|$ by correctness of $[\![\_]\!]$. Besides, if $e' = t \in \mathit{CTerm}$ is total then it is trivial to check that $|t| \sqsubseteq |e''|$ implies $t = |e''|$ thus getting $e'' = t$. □