Logical Methods in Computer Science
Vol. 13(4:16)2017, pp. 1–54
www.lmcs-online.org

Submitted May 31, 2016
Published Nov. 28, 2017

FAITHFUL (META-)ENCODINGS OF PROGRAMMABLE STRATEGIES INTO TERM REWRITING SYSTEMS *

HORATIU CIRSTEA, SERGUEÏ LENGLET, AND PIERRE-ETIENNE MOREAU

Université de Lorraine, CNRS, LORIA, UMR 7503, F-54506 Vandœuvre-lès-Nancy, France
e-mail address: {firstname.lastname@loria.fr}

Abstract. Rewriting is a formalism widely used in computer science and mathematical logic. When using rewriting as a programming or modeling paradigm, the rewrite rules describe the transformations one wants to operate and rewriting strategies are used to control their application. The operational semantics of these strategies are generally accepted and approaches for analyzing the termination of specific strategies have been studied. We propose in this paper a generic encoding of classic control and traversal strategies used in rewrite based languages such as Maude, Stratego and Tom into a plain term rewriting system. The encoding is proven sound and complete and, as a direct consequence, established termination methods used for term rewriting systems can be applied to analyze the termination of strategy controlled term rewriting systems. We show that the encoding of strategies into term rewriting systems can be easily adapted to handle many-sorted signatures and we use a meta-level representation of terms to reduce the size of the encodings. The corresponding implementation in Tom generates term rewriting systems compatible with the syntax of termination tools such as AProVE and TTT2, tools which turned out to be very effective in (dis)proving the termination of the generated term rewriting systems. The approach can also be seen as a generic strategy compiler which can be integrated into languages providing pattern matching primitives; experiments in Tom show that applying our encoding leads to performances comparable to the native Tom strategies.

1. Introduction

Term rewriting is a very powerful tool used in theoretical studies as well as for practical implementations. It is used, for example, in order to describe the meaning of programming languages [30, 35], and also to describe by inference rules a logic [19], a theorem prover [25], or a constraint solver [24]. Besides its use for specifying and implementing such formalisms, it is also used as an underlying computation mechanism in systems like Mathematica [42], Maude [9], or Tom [31, 4], where rewrite rules are objects of the language: they can be defined by the user and manipulated by other constructs.

Rewrite rules, the core concept in term rewriting, consist of a pattern that describes a schematic situation and the transformation that should be applied in that particular case.

Key words and phrases: Programmable strategies, termination, term rewriting systems.

* This is an extension of “A faithful encoding of programmable strategies into term rewriting systems” presented at RTA 2015.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-13(4:16)2017

For example, the following set of rewrite rules specify how to transform a Boolean expression into its corresponding disjunctive normal form (DNF):

$$\begin{array}{l}\mathtt{and}(\mathtt{or}(x,y),z) \twoheadrightarrow \mathtt{or}(\mathtt{and}(x,z),\mathtt{and}(y,z))\\ \mathtt{and}(z,\mathtt{or}(x,y)) \twoheadrightarrow \mathtt{or}(\mathtt{and}(z,x),\mathtt{and}(z,y))\end{array}$$

The left-hand side of each rule, *i.e.* its pattern, expresses a potentially infinite number of expressions on which the rule can be applied; these expressions are obtained by instantiating the variables x, y, z with arbitrary Boolean expressions. The application of the rewrite rule is decided using a (matching) algorithm which only depends on the pattern and on the term it is applied on, and not on the context in which it is applied. Given a term, a rewrite rule can be thus potentially applied on any of its sub-terms. The expression $\mathtt{and}(\mathtt{or}(\mathtt{P},\mathtt{Q}),\mathtt{and}(\mathtt{or}(\mathtt{P},\mathtt{Q}),\mathtt{R}))$ where $\mathtt{P}$, $\mathtt{Q}$, $\mathtt{R}$ are atoms (constants) of the language, is transformed either into $\mathtt{or}(\mathtt{and}(\mathtt{P},\mathtt{and}(\mathtt{or}(\mathtt{P},\mathtt{Q}),\mathtt{R})),\mathtt{and}(\mathtt{Q},\mathtt{and}(\mathtt{or}(\mathtt{P},\mathtt{Q}),\mathtt{R})))$, if we applied the first rule on the whole term, or into $\mathtt{and}(\mathtt{or}(\mathtt{P},\mathtt{Q}),\mathtt{or}(\mathtt{and}(\mathtt{P},\mathtt{R}),\mathtt{and}(\mathtt{Q},\mathtt{R})))$ if we apply it on the $\mathtt{and}$ sub-term. The latter term can be further reduced in two different ways depending on whether we apply the first or the second rule. Such sets of rewrite rules, called *term rewriting systems* (TRS), are thus very convenient for describing schematically the transformations one wants to operate.

In many situations, the application of a set of rewrite rules to a term eventually leads to the same final result independently on the way the rules are applied, and in such cases we say that the rewrite rules are *confluent* and *terminating*. This is for example the case for the above two rules. When using rewriting as a programming or modeling paradigm, it is nevertheless common to consider TRS that are non-confluent or non-terminating. For example, if we want to compute not only DNF but also conjunctive normal forms the following two rules could be added to the previous TRS:

$$\begin{array}{l}\mathtt{or}(\mathtt{and}(x,y),z) \twoheadrightarrow \mathtt{and}(\mathtt{or}(x,z),\mathtt{or}(y,z))\\ \mathtt{or}(z,\mathtt{and}(x,y)) \twoheadrightarrow \mathtt{and}(\mathtt{or}(z,x),\mathtt{or}(z,y))\end{array}$$

The resulting TRS is clearly non-terminating since we can have an infinite number of rule applications: $\mathtt{or}(\mathtt{and}(\mathtt{P},\mathtt{Q}),\mathtt{R})$ rewrites into $\mathtt{and}(\mathtt{or}(\mathtt{P},\mathtt{R}),\mathtt{or}(\mathtt{Q},\mathtt{R}))$, which in turn rewrites into $\mathtt{or}(\mathtt{and}(\mathtt{P},\mathtt{or}(\mathtt{Q},\mathtt{R})),\mathtt{and}(\mathtt{R},\mathtt{or}(\mathtt{Q},\mathtt{R})))$, and we can go on like this forever. The problem clearly comes here from the interference between the two sets of rules and there are several solutions to tackle this problem [40].

In a programming context, we can imagine that these rules are separated in different modules; this solves the termination problem but we could end up with a multitude of small modules if the problem occurs for several sets of rules. Another more general solution is functionalization [40] which consists in introducing some new *function symbols* and use them to implicitly guide the application of the rewrite rules modified accordingly. If we use now the symbol $\mathtt{at}$ to represent atoms, we can compute the DNF of a Boolean expression e by applying the following TRS on $\mathtt{DNF}(e)$:

$$\begin{array}{l}\mathtt{DNF}(\mathtt{at}(x)) \twoheadrightarrow \mathtt{at}(x)\\ \mathtt{DNF}(\mathtt{and}(x,y)) \twoheadrightarrow \mathtt{DNFr}(\mathtt{and}(\mathtt{DNF}(x),\mathtt{DNF}(y)))\\ \mathtt{DNF}(\mathtt{or}(x,y)) \twoheadrightarrow \mathtt{or}(\mathtt{DNF}(x),\mathtt{DNF}(y))\\ \mathtt{DNFr}(\mathtt{and}(\mathtt{or}(x,y),z)) \twoheadrightarrow \mathtt{or}(\mathtt{DNF}(\mathtt{and}(x,z)),\mathtt{DNF}(\mathtt{and}(y,z)))\\ \mathtt{DNFr}(\mathtt{and}(z,\mathtt{or}(x,y))) \twoheadrightarrow \mathtt{or}(\mathtt{DNF}(\mathtt{and}(z,x)),\mathtt{DNF}(\mathtt{and}(z,y)))\\ \mathtt{DNFr}(\mathtt{and}(\mathtt{at}(x),\mathtt{at}(y))) \twoheadrightarrow \mathtt{and}(\mathtt{at}(x),\mathtt{at}(y))\end{array}$$

The rules for `DNF` simulate the innermost normalization strategy by recursively traversing terms while the rules for `DNFr` are used to encode the distribution. However, for each new transformation, like, *e.g.*, computing the conjunctive normal forms, new symbols should be introduced and new rules for these symbols should be defined. Moreover, if we extend our grammar of Boolean expressions with new symbols, like for example negation `not` and implication `imp`, then new rewrite rules should be added for traversing the corresponding terms or for defining the base cases. Thus, this approach not only leads to a TRS with considerably more rules than the original one, but they are also more difficult to understand and to reuse.

The functionalization approach encodes an order on the application of rules and gives an explicit specification on how to traverse symbols. *Rewriting strategies* allow one to specify which rules should be applied and where in the term without having to write explicitly in the rewrite rules how to perform this control. Rule-based languages like Elan [6], Maude, Stratego [39], or Tom provide elementary rewriting strategies and allow the programmer to define its own strategies thanks to a specific strategy language. These user-defined strategies are usually called *programmable strategies* [40]. All these languages thus clearly distinguish between what we want to transform (the data structures), how to transform them (the rewrite rules), and how to control the application of these transformations (the programmable strategies).

Rewriting strategies may or may not change the semantics of the TRS they are controlling. If the TRS is terminating and confluent, imposing a reduction strategy would always lead to the same result, but the number of alternative reductions to get to this result is possibly smaller than for the uncontrolled TRS. Imposing an order on the (application of) the rules may reduce the number of alternative reductions even further. For example, if we extend the TRS computing the DNF with some simplification rules for Boolean expressions, we could give a higher priority to the application of some specific rules like, *e.g.* $\mathtt{and}(x, \mathtt{false}) \twoheadrightarrow \mathtt{false}$, to obtain shorter reductions. When the underlying TRS is non-confluent or non-terminating, the rewriting strategy could change its original semantics. The TRS consisting of the first four rules in this section is non-terminating, but when controlled by a strategy which applies in an innermost way only the first two rules, the termination is retrieved. Rewriting strategies can be also used to specify some default rules which are applied only when the others are not applicable. For example, if we replace the last rule in the TRS defining `DNF` with the more general rule $\mathtt{DNFr}(\mathtt{and}(x, y)) \twoheadrightarrow \mathtt{and}(x, y)$ the TRS would become non-confluent, but we can recover confluence by using a strategy which prioritizes the application of the other, meaningful rules, and gives the lowest priority to the last one.

Similarly to plain TRS (*i.e.* TRS without strategy), it is interesting to guarantee that a strategy-controlled TRS enjoys properties such as confluence and termination. Confluence holds as long as the rewrite rules are deterministic (*i.e.* the corresponding matching algorithm exhibits at most one solution) and all strategy operators are deterministic (or a deterministic implementation is provided). Confluence is clearly lost when considering non-deterministic strategies. Termination is more delicate and the normalization under some specific strategy is usually guaranteed by imposing (sufficient) conditions on the rewrite rules on which the strategy is applied. Such conditions have been proposed for the innermost [2, 15, 38, 20], outermost [10, 37, 20, 34], context-sensitive [16, 1, 20], or lazy [17] reduction strategies. Termination under programmable strategies has been studied for Elan [13] and Stratego [26, 28]. In [13], the authors give sufficient conditions for the termination of certain programmable strategies relying only on the rewrite rules involved in the strategy. Since the proposed criterion is applicable only to a specific subclass of programmable strategies and relies on a coarse-grained description of strategies, the approach cannot be used to prove termination for many terminating strategies. In [26, 28], the termination of some traversal strategies (such as top-down, bottom-up, innermost) is proven, assuming the rewrite rules are measure decreasing for a notion of measure that combines the depth and the number of occurrences of a specific constructor in a term.

Contributions. In this paper we describe a general approach consisting in translating programmable strategies into plain TRS. The interest of this encoding that we show sound and complete is twofold. First, termination analysis techniques [2, 22, 18] and corresponding tools like AProVE [14] and TTT2 [27] that have been successfully used for checking the termination of plain TRS can be used to verify termination in presence of rewriting strategies. Confluence can be analyzed in a similar way. Second, the translation can be seen as a generic strategy compiler and thus can be used as a portable implementation of strategies which could be easily integrated in any language accepting a term representation for the objects it manipulates and providing rewrite rules or at least pattern matching primitives.

This translation was introduced in [8]. We propose here two additional translations which are intended to improve the efficiency and expressiveness of the approach. The number of rewrite rules in the original encoding strongly depends on the considered signature; we introduce a meta-level representation of terms allowing to abstract over the signature leading thus to encodings whose number of rules is independent of the signature. We also consider many-sorted signatures and we show how the original translation handling only mono-sorted signatures can be adapted to generate well-sorted encodings executable in languages offering this feature. In summary, given a strategy S, we present an unsorted translation, a many-sorted one, as well as one working at meta-level, and we show that they produce faithful, *i.e.* sound and complete, encodings of the original strategy.

The translations have been implemented in Tom and generate TRS which could be fed into TTT2/AProVE for termination analysis or executed efficiently by Tom.

The paper is organized as follows. The next section introduces the notions of rewriting system and rewriting strategy. Section 3 presents the translation of rewriting strategies into rewriting systems (introduced in [8]), and in Sections 4 and 5, we describe the translations for meta-level terms and sorted terms, respectively. In Section 6, we give some implementation details and present experimental results. In the appendix, we provide detailed proofs of the properties stated in the paper and, in particular, of the simulation theorem from [8].

2. Strategic rewriting

This section briefly recalls some basic notions of rewriting used in this paper; see [3, 36] for more details on first order terms and term rewriting systems, and [41, 5] for details on rewriting strategies and their implementation in rewrite based languages.

2.1. **Term rewriting systems.** A *signature* Σ consists of a finite set $\mathcal{F}$ of symbols together with a function ar which associates to any symbol f its *arity*. We write $\mathcal{F}^n$ for the subset of symbols of arity n, and $\mathcal{F}^+$ for the symbols of arity $n > 0$. Symbols in $\mathcal{F}^0$ are called *constants*. Given a countable set $\mathcal{X}$ of *variable* symbols, the set of *first-order terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the smallest set containing $\mathcal{X}$ and such that $f(t_1, \ldots, t_n)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ whenever $f \in \mathcal{F}^n$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for $i \in [1, n]$. We write $\mathcal{V}ar\,(t)$ for the set of variables occuring in $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

If $\mathcal{V}ar\,(t)$ is empty, t is called a *ground* term; $\mathcal{T}(\mathcal{F})$ denotes the set of all ground terms. A *linear* term is a term where every variable occurs at most once. A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$ which is the identity except over a finite set of variables (its *domain*). A substitution extends as expected to a mapping from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$.

A *position* of a term t is a finite sequence of positive integers describing the path from the root of t to the root of the sub-term at that position. We write ε for the empty sequence, which represents the root of t, $\mathcal{P}os(t)$ for the set of positions of t, and $t_{|\omega}$ for the sub-term of t at position $\omega \in \mathcal{P}os(t)$. Finally, $t\,[s]_\omega$ is the term t with the sub-term at position ω replaced by s.

A *rewrite rule* (over Σ) is a pair $(l,r) \in \mathcal{T}(\mathcal{F},\mathcal{X}) \times \mathcal{T}(\mathcal{F},\mathcal{X})$ (also denoted $l \twoheadrightarrow r$) such that $\mathcal{V}ar\,(r) \subseteq \mathcal{V}ar\,(l)$ and a TRS is a set of rewrite rules $\mathcal{R}$ inducing a *rewriting relation* over $\mathcal{T}(\mathcal{F})$, denoted by $\longrightarrow_{\mathcal{R}}$ and such that $t \longrightarrow_{\mathcal{R}} t'$ iff there exist $l \twoheadrightarrow r \in \mathcal{R}$, $\omega \in \mathcal{P}os(t)$, and a substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_\omega$. In this case, $t_{|\omega}$ is a redex, l matches $t_{|\omega}$, and σ is the solution of the corresponding matching problem. The reflexive and transitive closure of $\longrightarrow_{\mathcal{R}}$ is denoted by $\longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}}$. In what follows, we generally use the notation $\mathcal{R} \bullet t \longrightarrow\!\!\!\!\!\rightarrow t'$ to denote $t \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}} t'$. A TRS $\mathcal{R}$ is *terminating* if there exists no infinite rewriting sequence $t_1 \longrightarrow_{\mathcal{R}} t_2 \longrightarrow_{\mathcal{R}} \ldots \longrightarrow_{\mathcal{R}} t_n \longrightarrow_{\mathcal{R}} \ldots$

2.2. **Rewriting strategies.** Rewriting strategies allow one to specify how rules should be applied. Classical strategies like innermost or outermost are considered often when implementing or reasoning about strategy controlled rewriting, but it could be useful to have more fine-grained strategies to control the application of rewrite rules. For example, we sometimes need to apply a set of rules only once, without retrying to apply the rules on the obtained result as with an innermost strategy. Consider a rule $\texttt{var}(x) \twoheadrightarrow \texttt{fresh}(\texttt{var}(x))$, whose purpose is to indicate that every variable in the abstract syntax tree of a program should be replaced by a corresponding fresh variable; to mark all the concerned variables it is sufficient to apply the rule once on the corresponding leaves of the tree, and it is needless to try the rule on upper positions in the tree or on the newly generated fresh variable. We could then combine this strategy with another one in charge of handling these fresh variables.

Taking the same terminology as the one proposed by Elan and Stratego, a rewrite rule is considered to be an elementary strategy, and a strategy is an expression built over a strategy language.

The strategy language we consider in this paper consists of the main operators used in Tom, Elan, and Stratego. Let Σ be a signature and $\mathcal{X}_{\mathcal{S}}$ be a set of strategy variables, ranged over by X. In what follows, we use uppercase for strategy variables and lowercase for term variables. We define the strategy language over Σ as follows:

$$S \ ::= \ \mathit{Identity} \mid \mathit{Fail} \mid l \twoheadrightarrow r \mid S; S \mid S \leftarrow\!\!\!\!+ S \mid \mathit{One}(S) \mid \mathit{All}(S) \mid \mu X \,.\, S \mid X$$

where $l \twoheadrightarrow r$ is any rewrite rule over Σ. We call the term to which the strategy is applied the subject.

The recursion operator $\mu X \,.\, S$ binds X in S; a strategy variable is said to be free if it is not bound. We write $\mathcal{FV}ar_{\mathcal{X}}\,(S)$ for the set of free variables of S. As usual, we work modulo *α-conversion* and we adopt Barendregt's *"hygiene-convention"*, *i.e.* free and bound variables have different names.

Informally, the *Identity* strategy can be applied to any term without changing it, and thus *Identity* always succeeds. Conversely, the strategy *Fail* always fails when applied to a term. As mentioned above, a rewrite rule is an elementary strategy which is (successfully or not) applied at the root position of its subject. By combining these elementary strategies, more complex strategies can be built: we can apply sequentially two strategies, make a choice between the application of two strategies, apply a strategy to one or to all the immediate sub-terms of the subject, and apply recursively a strategy.

The application of a strategy to a subject may diverge because recursion, fail because of the strategy *Fail* has been used or because a rewrite rule cannot be applied, or return a (unique) result. We use the symbol `Fail` to signal failure, and we let u range over terms and `Fail`. We implement recursion using a context Γ which maps variables to strategies; its syntax is defined as $\Gamma ::= \oslash \mid \Gamma; X : S$. When writing a context $X_1 : S_1; \ldots; X_n : S_n$, we omit the empty context $\oslash$, and we assume that the variables $(X_i)_{1 \leq i \leq n}$ are pairwise distinct. We write $\mathcal{D}om_{\mathcal{X}}(\Gamma)$ for the domain of Γ, defined as $\mathcal{D}om_{\mathcal{X}}(\oslash) = \emptyset$, and $\mathcal{D}om_{\mathcal{X}}(\Gamma; X : S) = \mathcal{D}om_{\mathcal{X}}(\Gamma) \cup \{X\}$. We define the evaluation judgment $\Gamma \vdash S \circ t \Longrightarrow u$ inductively by the rules of Figure 1; it means that the application of the strategy S to the subject t produces the result u under the context Γ (the context may be omitted if empty). The semantics being defined inductively and in a big-step style, there is no derivation for $\Gamma \vdash S \circ t$ if the application of S to t diverges (*e.g.* if $S = \mu X \,.\, X$). There is also no derivation if a free variable X of S is applied to a term and $X \notin \mathcal{D}om_{\mathcal{X}}(\Gamma)$.

We distinguish between three kinds of operators in the strategy language:

- *elementary strategies* consisting of *Identity*, *Fail*, and rewrite rules, which are the basic building blocks for strategies;
- *control combinators* consisting of choice $S_1 \leftarrow\!\!\!+ S_2$, sequence $S_1; S_2$, and recursion $\mu X \,.\, S$, that compose strategies but are still applied at the root position of the subject;
- *traversal combinators* *One*(S) and *All*(S) that modify the current application position.

We stress that a rewrite rule taken as an elementary strategy is not applied at every possible position of a subject, but only at the root position, as we can see with the following example.

Example 2.1. Let Σ be the signature corresponding to Peano natural numbers such that $\mathcal{F}^0 = \{\mathtt{Z}\}$, $\mathcal{F}^1 = \{\mathtt{S}\}$, and $\mathcal{F}^2 = \{\mathtt{+}\}$, and consider the rewrite rule $\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x$. Then, we have $\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z})) \Longrightarrow \mathtt{S}(\mathtt{Z})$, but $\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{S}(\mathtt{+}(\mathtt{Z}, \mathtt{Z})) \Longrightarrow \mathtt{Fail}$, because the rule cannot be applied at the root position in the second case. Similarly, only the redex at the root position is reduced in $\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))$.

Control combinators are also applied at the root position. The (left-)choice $S_1 \leftarrow\!\!\!+ S_2$ tries to apply S_1 and considers S_2 only if S_1 fails. Using this operator, we can then define a strategy *Try*(S) which tries to apply S and applies the identity if S fails: *Try*$(S) = S \leftarrow\!\!\!+$ *Identity*. Given a set of rules $R_1, \ldots, R_n$, the strategy $R_1 \leftarrow\!\!\!+ (\cdots \leftarrow\!\!\!+ R_n)$ can be used to express an order on the rules, mirroring how pattern matching is done in most functional programming languages. The sequential application $S_1; S_2$ succeeds if S_1 succeeds on the subject and S_2 succeeds on the subsequent term; it fails if one of the two strategy applications fails. We consider these two operators to be right-associative, so that $S_1 \leftarrow\!\!\!+ (S_2 \leftarrow\!\!\!+ S_3)$ is written $S_1 \leftarrow\!\!\!+ S_2 \leftarrow\!\!\!+ S_3$ and $S_1; (S_2; S_3)$ is written $S_1; S_2; S_3$.

Example 2.2. We can apply sequentially twice the rewrite rule $\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x$ using the strategy $\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x; \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x$. As we have seen in Example 2.1, the rule $\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x$ applied once to $\mathtt{+}(\mathtt{Z}, \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z})))$ gives $\mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))$, and since the rule can then be applied again at the root position, we eventually obtain $\mathtt{S}(\mathtt{Z})$ as a final result:

$$\dfrac{\dfrac{}{\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))}\,(\mathbf{r_1}) \quad \dfrac{}{\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z})) \Longrightarrow \mathtt{S}(\mathtt{Z})}\,(\mathbf{r_1})}{\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x; \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{S}(\mathtt{Z})}\,(\mathbf{seq_1})$$

Elementary strategies

$$\frac{}{\Gamma \vdash \mathit{Identity} \circ t \Longrightarrow t}\ (\mathbf{id}) \qquad \frac{}{\Gamma \vdash \mathit{Fail} \circ t \Longrightarrow \mathtt{Fail}}\ (\mathbf{fail}) \qquad \frac{\exists \sigma, \sigma(l) = t}{\Gamma \vdash l \twoheadrightarrow r \circ t \Longrightarrow \sigma(r)}\ (\mathbf{r_1})$$

$$\frac{\nexists \sigma, \sigma(l) = t}{\Gamma \vdash l \twoheadrightarrow r \circ t \Longrightarrow \mathtt{Fail}}\ (\mathbf{r_2})$$

Control combinators

$$\frac{\Gamma \vdash S_1 \circ t \Longrightarrow t'}{\Gamma \vdash (S_1 \leftarrow\!\!\!+ S_2) \circ t \Longrightarrow t'}\ (\mathbf{choice_1}) \qquad \frac{\Gamma \vdash S_1 \circ t \Longrightarrow \mathtt{Fail} \qquad \Gamma \vdash S_2 \circ t \Longrightarrow u}{\Gamma \vdash (S_1 \leftarrow\!\!\!+ S_2) \circ t \Longrightarrow u}\ (\mathbf{choice_2})$$

$$\frac{\Gamma \vdash S_1 \circ t \Longrightarrow t' \qquad \Gamma \vdash S_2 \circ t' \Longrightarrow u}{\Gamma \vdash (S_1; S_2) \circ t \Longrightarrow u}\ (\mathbf{seq_1}) \qquad \frac{\Gamma \vdash S_1 \circ t \Longrightarrow \mathtt{Fail}}{\Gamma \vdash (S_1; S_2) \circ t \Longrightarrow \mathtt{Fail}}\ (\mathbf{seq_2})$$

$$\frac{\Gamma; X : S \vdash S \circ t \Longrightarrow u}{\Gamma \vdash \mu X \,.\, S \circ t \Longrightarrow u}\ (\mathbf{mu}) \qquad \frac{\Gamma; X : S \vdash S \circ t \Longrightarrow u}{\Gamma; X : S \vdash X \circ t \Longrightarrow u}\ (\mathbf{muvar})$$

Traversal combinators

$$\frac{\exists i \in [1, n], (\Gamma \vdash S \circ t_i \Longrightarrow t'_i \wedge \forall j \in [1, i-1], \Gamma \vdash S \circ t_j \Longrightarrow \mathtt{Fail})}{\Gamma \vdash \mathit{One}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n)}\ (\mathbf{one_1})$$

$$\frac{\forall i \in [1, n], \Gamma \vdash S \circ t_i \Longrightarrow \mathtt{Fail}}{\Gamma \vdash \mathit{One}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow \mathtt{Fail}}\ (\mathbf{one_2})$$

$$\frac{\forall i \in [1,n], \Gamma \vdash S \circ t_i \Longrightarrow t'_i}{\Gamma \vdash \mathit{All}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow f(t'_1, \ldots, t'_n)}\ (\mathbf{all_1})$$

$$\frac{\exists i \in [1,n], \Gamma \vdash S \circ t_i \Longrightarrow \mathtt{Fail}}{\Gamma \vdash \mathit{All}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow \mathtt{Fail}}\ (\mathbf{all_2})$$

Figure 1: Strategy semantics; t and its indexed and primed versions denote terms (which cannot be $\mathtt{Fail}$), whereas u denotes a result which is either a well-formed term, or $\mathtt{Fail}$.

However, the application of the same strategy on the term $\mathtt{+(Z, S(+(Z, Z)))}$ fails because the first application produces $\mathtt{S(+(Z, Z))}$, on which we cannot apply the rule again:

$$\frac{\dfrac{}{\vdash \mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x \circ \mathtt{+(Z, S(+(Z, Z)))} \Longrightarrow \mathtt{S(+(Z, Z))}}\ (\mathbf{r_1}) \quad \dfrac{}{\vdash \mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x \circ \mathtt{S(+(Z, Z))} \Longrightarrow \mathtt{Fail}}\ (\mathbf{r_2})}{\vdash \mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x; \mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x \circ \mathtt{+(Z, S(+(Z, Z)))} \Longrightarrow \mathtt{Fail}}\ (\mathbf{seq_2})$$

We can avoid failure using the strategy $\mathit{Try}(\mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x; \mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x)$ which represents in fact the strategy $(\mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x; \mathtt{+(Z,}x\mathtt{)} \twoheadrightarrow x) \leftarrow\!\!\!+ \mathit{Identity}$. If we denote T the above derivation tree, we obtain the following evaluation when applying the Try strategy:

$$\dfrac{T \qquad \dfrac{}{\vdash \mathit{Identity} \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{+}(\texttt{Z},\texttt{Z}))) \Longrightarrow \texttt{+}(\texttt{Z},\texttt{S}(\texttt{+}(\texttt{Z},\texttt{Z})))}\,(\textbf{id})}{\vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; \texttt{+}(\texttt{Z},x) \rightarrowtriangle x) \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{+}(\texttt{Z},\texttt{Z}))) \Longrightarrow \texttt{+}(\texttt{Z},\texttt{S}(\texttt{+}(\texttt{Z},\texttt{Z})))}\,(\textbf{choice}_\mathbf{2})$$

Note that the resulting term is the initial subject, not the intermediate term $\texttt{S}(\texttt{+}(\texttt{Z},\texttt{Z}))$ responsible for the failure. To obtain this intermediate term, we should have applied $\mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x); \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x)$ to the subject.

The application of a recursive strategy $\mu X \,.\, S$ to a subject t applies S to t in a context where X is mapped to S (rule **mu**). During this evaluation, the strategy variable X may be applied to a term t', triggering the application of S to t' (rule **muvar**), allowing thus recursion. For the recursion to stop, the evaluation of S should not involve X anymore at some point, otherwise the application of $\mu X.S$ to t diverges. For example, the strategy $\mu X.X$ always diverges when applied to any subject, and we cannot derive any evaluation judgment for it. A more interesting example of recursive strategy is $\mathit{Repeat}(S) = \mu X \,.\, \mathit{Try}(S; X)$ which applies a strategy S as much as possible at the root position of the subject. The last successful result of the application of S is the overall result of $\mathit{Repeat}(S)$. Note that applying a *Repeat* strategy either diverges or evaluates to a term (and not to `Fail`).

Example 2.3. To see how recursion works, let us build the derivation tree for the application of the strategy $\mathit{Repeat}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x) = \mu X \,.\, \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X)$ to the term $t = \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})))$. To evaluate this application we necessarily start by applying the rule (**mu**), which adds to the context the mapping of X to $\mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X)$ and triggers the evaluation of this latter strategy. Thus, supposing the derivation tree can be eventually built then, it has the shape

$$\dfrac{\dfrac{\vdots}{X : \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow u}}{\vdash \mu X \,.\, \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow u}\,(\textbf{mu})$$

The subsequent evaluations in this derivation tree are performed *w.r.t.* this context $\Gamma_{pz} = X : \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X)$. In order to apply the strategy $\mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) = \texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X \leftarrow\!\!\!\!\!+ \mathit{Identity}$ to t, we have to decide between the two **choice** rules; in both cases, we have to apply the strategy $\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X$, and subsequently the rewrite rule $\texttt{+}(\texttt{Z},x) \rightarrowtriangle x$, to t. This rewrite rule succeeds when applied at the root position of t, and thus the derivation tree has the form:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_{pz} \vdash \texttt{+}(\texttt{Z},x) \rightarrowtriangle x \circ \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))}\,(\mathbf{r_1}) \qquad \dfrac{T_1}{\Gamma_{pz} \vdash X \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})) \Longrightarrow u}\,(\textbf{muvar})}{\Gamma_{pz} \vdash \texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X \circ \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow u}\,(\textbf{seq}_\mathbf{1})}{\Gamma_{pz} \vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow u}\,(\textbf{choice}_\mathbf{1})}{\vdash \mu X \,.\, \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow u}\,(\textbf{mu})$$

with T_1 the derivation tree for the judgment $\Gamma_{pz} \vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})) \Longrightarrow u$ obtained by instantiating the variable X with the corresponding strategy from the context. We can apply the same reasoning as above, and since the rewrite rule $\texttt{+}(\texttt{Z},x) \rightarrowtriangle x$ also succeeds on the new term $\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))$, the derivation tree T_1 has the following shape:

$$\dfrac{\dfrac{\dfrac{}{\Gamma_{pz} \vdash \texttt{+}(\texttt{Z},x) \rightarrowtriangle x \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})) \Longrightarrow \texttt{S}(\texttt{Z})}\,(\mathbf{r_1}) \qquad \dfrac{T_2}{\Gamma_{pz} \vdash X \circ \texttt{S}(\texttt{Z}) \Longrightarrow u}\,(\mathbf{muvar})}{\Gamma_{pz} \vdash \texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})) \Longrightarrow u}\,(\mathbf{seq_1})}{\Gamma_{pz} \vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})) \Longrightarrow u}\,(\mathbf{choice_1})$$

with T_2 the derivation tree for the judgment $\Gamma_{pz} \vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{S}(\texttt{Z}) \Longrightarrow u$ obtained, as before, by instantiating accordingly the variable X. The rewrite rule $\texttt{+}(\texttt{Z},x) \rightarrowtriangle x$ fails when applied to the term $\texttt{S}(\texttt{Z})$ and the derivation tree T_2 is therefore as follows:

$$\dfrac{\dfrac{\dfrac{}{\Gamma_{pz} \vdash \texttt{+}(\texttt{Z},x) \rightarrowtriangle x \circ \texttt{S}(\texttt{Z}) \Longrightarrow \texttt{Fail}}\,(\mathbf{r_2})}{\Gamma_{pz} \vdash \texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X \circ \texttt{S}(\texttt{Z}) \Longrightarrow \texttt{Fail}}\,(\mathbf{seq_2}) \qquad \dfrac{}{\Gamma_{pz} \vdash \mathit{Identity} \circ \texttt{S}(\texttt{Z}) \Longrightarrow \texttt{S}(\texttt{Z})}\,(\mathbf{id})}{\Gamma_{pz} \vdash \mathit{Try}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x; X) \circ \texttt{S}(\texttt{Z}) \Longrightarrow \texttt{S}(\texttt{Z})}\,(\mathbf{choice_2})$$

This completes the derivation tree, and we can thus conclude that the result of applying the strategy $\mathit{Repeat}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x)$ to the term $\texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})))$ is $u = \texttt{S}(\texttt{Z})$.

So far, the strategies are applied only at the root position; we use traversal combinators to move to other positions. The strategy $\mathit{One}(S)$ tries to apply S to a sub-term of the subject, starting from the leftmost one (rule $\mathbf{one_1}$). If S fails on all the sub-terms, then $\mathit{One}(S)$ also fails (rule $\mathbf{one_2}$). In contrast, $\mathit{All}(S)$ applies S to all the sub-terms of the subject (rule $\mathbf{all_1}$) and fails if S fails on one of them (rule $\mathbf{all_2}$). Note that $\mathit{One}(S)$ always fails when applied to a constant while $\mathit{All}(S)$ always succeeds in this case.

Example 2.4. In $\vdash \mathit{One}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x) \circ \texttt{+}(\texttt{+}(\texttt{Z},\texttt{Z}),\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow \texttt{+}(\texttt{Z},\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z})))$, only the first sub-term is reduced, while in $\vdash \mathit{All}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x) \circ \texttt{+}(\texttt{+}(\texttt{Z},\texttt{Z}),\texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))) \Longrightarrow \texttt{+}(\texttt{Z},\texttt{S}(\texttt{Z}))$, they are both reduced. These operators give access only to the immediate sub-term: we have $\vdash \mathit{One}(\texttt{+}(\texttt{Z},x) \rightarrowtriangle x) \circ \texttt{S}(\texttt{S}(\texttt{+}(\texttt{Z},\texttt{Z}))) \Longrightarrow \texttt{Fail}$, and similarly with All.

Combined with recursion, traversal combinators can be used to have access to any position of a term. In fact, most of the classic reduction strategies can be defined using recursion and traversal operators:

$$\begin{array}{rcl} \mathit{OnceBottomUp}(S) & = & \mu X \,.\, \mathit{One}(X) \nleftarrow S \\ \mathit{BottomUp}(S) & = & \mu X \,.\, \mathit{All}(X); S \\ \mathit{OnceTopDown}(S) & = & \mu X \,.\, S \nleftarrow \mathit{One}(X) \\ \mathit{TopDown}(S) & = & \mu X \,.\, S; \mathit{All}(X) \end{array}$$

OnceBottomUp (denoted *obu* in the following) tries to apply a strategy S once, starting from the leftmost-innermost leaves. *BottomUp* behaves almost like *obu* except that S is applied to all nodes, starting from the leaves. *OnceTopDown* and *TopDown* are similar, except they start from the root position. The strategy which applies S as many times as possible, starting from the leaves can be either defined naively as $\mathit{Repeat}(\mathit{OnceBottomUp}(S))$ or using a more efficient approach [41]: $\mathit{Innermost}(S) = \mu X \,.\, \mathit{All}(X); \mathit{Try}(S; X)$.

Example 2.5. We use *OnceTopDown*, *OnceBottomUp*, and *Innermost*, when applying $S = \texttt{+}(\texttt{Z},x) \rightarrowtriangle x \nleftarrow \texttt{+}(\texttt{S}(x),y) \rightarrowtriangle \texttt{S}(\texttt{+}(x,y))$ to $t = \texttt{+}(\texttt{+}(\texttt{Z},\texttt{Z}),\texttt{+}(\texttt{S}(\texttt{Z}),\texttt{Z}))$. With $\mathit{OnceTopDown}(S)$, we start at the root position of t and since there is no redex at this position, we recursively try to apply S to one of its children: we can apply it on the first child and obtain $\texttt{+}(\texttt{Z},\texttt{+}(\texttt{S}(\texttt{Z}),\texttt{Z}))$. If we apply $\mathit{OnceTopDown}(S)$ on this latter term we obtain $\texttt{+}(\texttt{S}(\texttt{Z}),\texttt{Z})$. With

$\mathit{OnceBottomUp}(S)$, we start from the leaves, and we obtain the same term $\texttt{+}(\texttt{Z}, \texttt{+}(\texttt{S}(\texttt{Z}), \texttt{Z}))$ as before but if we apply $\mathit{OnceBottomUp}(S)$ once again on this term we obtain now $\texttt{+}(\texttt{Z}, \texttt{S}(\texttt{+}(\texttt{Z}, \texttt{Z})))$. $\mathit{Innermost}(S)$ also starts from the leaves but makes a recursive call after each application of S, so we obtain $\vdash \mathit{Innermost}(S) \circ t \Longrightarrow \texttt{S}(\texttt{Z})$.

Both $\mathit{BottomUp}(S)$ and $\mathit{TopDown}(S)$ fail on t but we have $\vdash \mathit{BottomUp}(\mathit{Try}(S)) \circ t \Longrightarrow \texttt{S}(\texttt{+}(\texttt{Z}, \texttt{Z}))$ and $\vdash \mathit{TopDown}(\mathit{Try}(S)) \circ t \Longrightarrow \texttt{+}(\texttt{Z}, \texttt{S}(\texttt{Z}))$.

Remark 2.6. The semantics of *One* is deterministic, as it looks for the leftmost sub-term that can be successfully transformed; a non-deterministic behavior can easily be obtained by removing the second condition in the premises of the inference rule $\mathbf{one_1}$. Similarly, we can adopt a non-deterministic semantics for $S_1 \leftarrow\!\!\!+ S_2$, where either S_1 or S_2 is tried first, by removing the first judgment in the premises of the inference rule $\mathbf{choice_2}$. We focus here on the encoding of the deterministic semantics but we also explain in the next section (Remark 3.4) how our encoding can be adapted to handle the non-deterministic versions of *One* and choice.

Given this semantics, we can define a notion of termination for strategies.

Definition 2.7. Let S, Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$. The strategy S is terminating under Γ if for all $t \in \mathcal{T}(\mathcal{F})$, there exists u such that $\Gamma \vdash S \circ t \Longrightarrow u$.

The condition on the variables of S and Γ ensures that if there is no derivation for $\Gamma \vdash S \circ t$, this is not because a free variable of S does not occur in the domain of Γ. If $\mathcal{FV}ar_{\mathcal{X}}(S) = \emptyset$, then Γ does not play a role in the derivation, and we simply say that S is terminating as a shorthand for S is terminating under any context. Note that we allow terminating strategies to fail, as u can be $\texttt{Fail}$. A terminating strategy may contain non-terminating rewrite rules: the strategy $\texttt{Z} \twoheadrightarrow \texttt{+}(\texttt{Z}, \texttt{Z})$ is terminating because the rule is applied once at the root position (and either succeeds or fails). However, $\mathit{Innermost}(\texttt{Z} \twoheadrightarrow \texttt{+}(\texttt{Z}, \texttt{Z}))$ is not terminating. A terminating strategy may also contain a non-terminating strategy: $\mathit{Identity} \leftarrow\!\!\!+ \mu X \,.\, X$ is terminating because the diverging part $\mu X \,.\, X$ is never applied.

3. Encoding rewriting strategies with rewrite rules

We translate a strategy into a faithful TRS, in the sense that a strategy applied to a term produces a result iff the TRS also rewrites the encoded term into the encoded result.

3.1. **Strategy translation.** The evaluation of the application of a strategy S on a term t consists in selecting and applying the corresponding sub-strategies of S to t when the head operator of S is a control combinator (*e.g.* selecting S_1 in $S_1 \leftarrow\!\!\!+ S_2$ in the inference rule $\mathbf{choice_1}$), in applying S on the corresponding sub-terms of t when the head operator of S is a traversal combinator (*e.g.* applying S on the sub-term t_i of $f(t_1, \ldots, t_n)$ in the inference rule $\mathbf{one_1}$), and eventually in applying elementary strategies. The translation function presented in Figure 2 associates to each strategy a set of rewrite rules which encodes exactly this behaviour and preserves the original evaluation: $\mathbb{T}(S) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow t'$ whenever $\vdash S \circ t \Longrightarrow t'$ (the exact relationship between a strategy and its encoding is formally stated in Section 3.2).

Since the application of the rewrite rules of the encoding cannot be explicitly controlled, the first issue is to ensure that only the rules of the appropriate (sub-)strategy can be applied to the current subject, and that an encoded strategy is applied to the right sub-term(s) in the case of traversal combinators. The other difficulties concern the representations of contexts Γ, as well as of (matching and strategy) failures, since these notions are not explicit in plain term rewriting. To deal with the first problem, we introduce some symbols φ used to implicitly control the application of the rewrite rules encoding a strategy. The set $\mathbb{T}(S)$ thus contains rules whose left-hand sides are headed by a symbol φ_S and which encode the behaviour of the strategy S by using, in the right-hand sides, the symbols φ corresponding to the sub-strategies of S and potentially some auxiliary symbols. All these symbols are supposed to be freshly generated and uniquely identified, *i.e.* there will be only one φ_S symbol for each encoded (sub-)strategy S and each auxiliary φ symbol can be identified by the strategy it has been generated for. For example, in the encoding $\mathbb{T}(S_1\,;S_2)$, the symbol $\varphi_;$ is just an abbreviation for $\varphi_;^{S_1;S_2}$, *i.e.* the specific $\varphi_;$ used for the encoding of the strategy $S_1\,;S_2$.

In addition to these φ symbols, we also use a particular symbol $\perp$ of arity 1 which encodes failure and whose argument is used, as explained later on in this section, to keep track of the term on which the strategy failed. For example, the fact that the strategy *Fail* applied to t results in `Fail` corresponds in our encoding to the reduction of $\varphi_{Fail}(t)$ *w.r.t.* $\mathbb{T}(Fail)$ into $\perp(t)$. This additional information attached to a failure is particularly helpful when encoding strategies where a sub-strategy is tried successively on different terms, namely for the choice and for *One*.

Apart from the application of *Fail*, strategy failures only come from matching failures. We have to be careful in encoding these failures since we should conclude that a matching failure occurred only if the terms involved are built over the initial signature and contain no generated symbols. For this we use the so-called *anti-terms*[1] of the form $!t$ with $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ or $!\perp(x)$ with $x \in \mathcal{X}$. An anti-term $!t$ represents all the ground terms in $\mathcal{T}(\mathcal{F})$ which do not match t and the anti-pattern $!\perp(x)$ denotes all the ground terms $t \in \mathcal{T}(\mathcal{F})$ of the original signature; the way the finite representation of these terms is generated is explained in Section 6. For example, if we consider the signature from Example 2.1, $!\texttt{+}(\texttt{Z}, x)$ denotes exactly the terms matched by `Z`, $\texttt{S}(x_1)$, $\texttt{+}(\texttt{S}(x_1), x_2)$, or $\texttt{+}(\texttt{+}(x_1, x_2), x_3)$. We should emphasize that in this encoding, the semantics of $!t$ is considered *w.r.t.* the terms in $\mathcal{T}(\mathcal{F})$. For example, $!c$ for some constant c does not include $\perp(c)$ or terms of the form $\varphi(t_1, \ldots, t_n)$, because $\perp$ and φ symbols do not belong to the original signature.

To keep the presentation of the translation compact and intuitive, we express it using rule schemas involving anti-terms and a special aliasing symbol "@": each rule in Figure 2 involving these notations represent in fact a set of rewrite rules. Anti-terms in the left-hand sides of rules can be aliased with @ by variables which can be then conveniently used in the right-hand sides of the corresponding rules. The rewrite rule $l \twoheadrightarrow r$ with l, r such that $l_{|\omega} = x \,@\, !u$ and $r_{|\omega'} = x$, for some $u \in \mathcal{T}(\mathcal{F},\mathcal{X}) \cup \{\perp(x) \mid x \in \mathcal{X}\}$ and positions ω, ω' is thus just an alias for the set of rewrite rules $l\,[u_1]_\omega \twoheadrightarrow r\,[u_1]_{\omega'}, \ldots, l\,[u_n]_\omega \twoheadrightarrow r\,[u_n]_{\omega'}$, where $u_1, \ldots, u_n$ are the terms represented by $!u$. Moreover, the variable symbol "_" can be used in the left-hand side of a rule to indicate a variable that does not appear in the right-hand side.

For example, the rule schema $\varphi(y \,@\, !\texttt{+}(\texttt{Z}, \_)) \twoheadrightarrow \perp(y)$ denotes the set of rewrite rules consisting of $\varphi(\texttt{Z}) \twoheadrightarrow \perp(\texttt{Z})$, $\varphi(\texttt{S}(y_1)) \twoheadrightarrow \perp(\texttt{S}(y_1))$, $\varphi(\texttt{+}(\texttt{S}(y_1), y_2)) \twoheadrightarrow \perp(\texttt{+}(\texttt{S}(y_1), y_2))$, and $\varphi(\texttt{+}(\texttt{+}(y_1, y_2), y_3)) \twoheadrightarrow \perp(\texttt{+}(\texttt{+}(y_1, y_2), y_3))$.

[1]We restrict here to a limited form of anti-terms; we refer to [7] for the complete semantics of anti-terms.

$$
\begin{array}{lrcl}
(\mathbf{E1}) & \mathbb{T}(\mathit{Identity}) & = & \{\ \varphi_{\mathit{Identity}}(x\,@\,!\bot(\_)) \twoheadrightarrow x, \quad \varphi_{\mathit{Identity}}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E2}) & \mathbb{T}(\mathit{Fail}) & = & \{\ \varphi_{\mathit{Fail}}(x\,@\,!\bot(\_)) \twoheadrightarrow \bot(x), \quad \varphi_{\mathit{Fail}}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E3}) & \mathbb{T}(l \twoheadrightarrow r) & = & \{\ \varphi_{l\twoheadrightarrow r}(l) \twoheadrightarrow r, \\
& & & \ \ \varphi_{l\twoheadrightarrow r}(x\,@\,!l) \twoheadrightarrow \bot(x), \quad \varphi_{l\twoheadrightarrow r}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\\
(\mathbf{E4}) & \mathbb{T}(S_1;S_2) & = & \mathbb{T}(S_1)\ \cup\ \mathbb{T}(S_2) \\
& & \bigcup & \{\ \varphi_{S_1;S_2}(x\,@\,!\bot(\_)) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)),x), \quad \varphi_{S_1;S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
& & & \ \ \varphi_{;}(x\,@\,!\bot(\_),\_) \twoheadrightarrow x, \quad \varphi_{;}(\bot(\_),x) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E5}) & \mathbb{T}(S_1 \leftarrow\!\!\!+ S_2) & = & \mathbb{T}(S_1)\ \cup\ \mathbb{T}(S_2) \\
& & \bigcup & \{\ \varphi_{S_1\leftarrow\!\!\!+ S_2}(x\,@\,!\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(x)), \quad \varphi_{S_1\leftarrow\!\!\!+ S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
& & & \ \ \varphi_{\leftarrow\!\!\!+}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x), \quad \varphi_{\leftarrow\!\!\!+}(x\,@\,!\bot(\_)) \twoheadrightarrow x\ \} \\
(\mathbf{E6}) & \mathbb{T}(\mu X\,.\,S) & = & \mathbb{T}(S) \\
& & \bigcup & \{\ \varphi_{\mu X.S}(x\,@\,!\bot(\_)) \twoheadrightarrow \varphi_S(x), \quad \varphi_{\mu X.S}(\bot(x)) \twoheadrightarrow \bot(x), \\
& & & \ \ \varphi_X(x\,@\,!\bot(\_)) \twoheadrightarrow \varphi_S(x), \quad \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E7}) & \mathbb{T}(X) & = & \emptyset \\
\\
(\mathbf{E8}) & \mathbb{T}(\mathit{All}(S)) & = & \mathbb{T}(S) \\
& & \bigcup & \{\ \varphi_{\mathit{All}(S)}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
& & \bigcup_{c\in\mathcal{F}^0} & \{\ \varphi_{\mathit{All}(S)}(c) \twoheadrightarrow c\ \} \\
& & \bigcup_{f\in\mathcal{F}^+} & \{\ \varphi_{\mathit{All}(S)}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_f(\varphi_S(x_1),\ldots,\varphi_S(x_n),f(x_1,\ldots,x_n)), \\
& & & \ \ \varphi_f(x_1\,@\,!\bot(\_),\ldots,x_n\,@\,!\bot(\_),\_) \twoheadrightarrow f(x_1,\ldots,x_n), \\
& & & \ \ \varphi_f(\bot(\_),\_,\ldots,\_,x) \twoheadrightarrow \bot(x), \\
& & & \ \ \vdots \\
& & & \ \ \varphi_f(\_,\ldots,\_,\bot(\_),x) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E9}) & \mathbb{T}(\mathit{One}(S)) & = & \mathbb{T}(S) \\
& & \bigcup & \{\ \varphi_{\mathit{One}(S)}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
& & \bigcup_{c\in\mathcal{F}^0} & \{\ \varphi_{\mathit{One}(S)}(c) \twoheadrightarrow \bot(c)\ \} \\
& & \bigcup_{f\in\mathcal{F}^+} & \{\ \varphi_{\mathit{One}(S)}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_{f_1}(\varphi_S(x_1),x_2,\ldots,x_n)\ \} \\
& \bigcup_{f\in\mathcal{F}^+} & \bigcup_{1\le i\le ar(f)} & \{\ \varphi_{f_i}(\bot(x_1),\ldots,\bot(x_{i-1}),x_i\,@\,!\bot(\_),x_{i+1},\ldots,x_n) \twoheadrightarrow f(x_1,\ldots,x_n)\} \\
& \bigcup_{f\in\mathcal{F}^+} & \bigcup_{1\le i< ar(f)} & \{\ \varphi_{f_i}(\bot(x_1),\ldots,\bot(x_i),x_{i+1},\ldots,x_n) \twoheadrightarrow \\
& & & \qquad\qquad \varphi_{f_{i+1}}(\bot(x_1),\ldots,\bot(x_i),\varphi_S(x_{i+1}),x_{i+2},\ldots,x_n)\ \} \\
& & \bigcup_{f\in\mathcal{F}^+} & \{\ \varphi_{f_n}(\bot(x_1),\ldots,\bot(x_n)) \twoheadrightarrow \bot(f(x_1,\ldots,x_n))\ \} \\
\\
(\mathbf{E10}) & \mathbb{B}(\Gamma;X\colon S) & = & \mathbb{B}(\Gamma)\ \cup\ \mathbb{T}(S) \\
& & \bigcup & \{\ \varphi_X(x\,@\,!\bot(\_)) \twoheadrightarrow \varphi_S(x), \quad \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E11}) & \mathbb{B}(\oslash) & = & \emptyset
\end{array}
$$

Figure 2: Strategy translation.

The translation of the *Identity* strategy (**E1**) consists of a rule whose left-hand side matches any term in the signature[2] (contextualized by the corresponding φ symbol) and whose right-hand side is the initial term, and of a rule encoding strict propagation of failure. This latter rule guarantees a faithful encoding of the strategy guided evaluation and is in fact present, in different forms, in the translations of all the strategy operators.

Example 3.1. If we consider the signature in Example 2.1, the following encoding is obtained for the *Identity* strategy:

$$\begin{aligned}\mathbb{T}(\mathit{Identity}) = \{\ & \varphi_{\mathit{Identity}}(\mathtt{Z}) \twoheadrightarrow \mathtt{Z},\\ & \varphi_{\mathit{Identity}}(\mathtt{S}(y_1)) \twoheadrightarrow \mathtt{S}(y_1),\\ & \varphi_{\mathit{Identity}}(\mathtt{+}(y_1,y_2)) \twoheadrightarrow \mathtt{+}(y_1,y_2),\\ & \varphi_{\mathit{Identity}}(\bot(x)) \twoheadrightarrow \bot(x)\ \}\end{aligned}$$

Similarly, the translation of the *Fail* strategy (**E2**) contains a failure propagation rule, and a rule whose left-hand side matches any term and whose right-hand side is a failure keeping track of this term. A rewrite rule (which is an elementary strategy applicable at the root of the subject) is translated (**E3**) by two rules encoding the behaviour in case of respectively a matching success or a failure, together with a rule for failure propagation.

Example 3.2. The strategy $S_{pz} = \mathtt{+}(\mathtt{Z},x) \twoheadrightarrow x$ is encoded by the following rules:

$$\begin{aligned}\mathbb{T}(S_{pz}) = \{\ & \varphi_{pz}(\mathtt{+}(\mathtt{Z},x)) \twoheadrightarrow x,\\ & \varphi_{pz}(y \mathbin{@} !\mathtt{+}(\mathtt{Z},x)) \twoheadrightarrow \bot(y),\\ & \varphi_{pz}(\bot(x)) \twoheadrightarrow \bot(x)\ \}\end{aligned}$$

which lead, when the anti-terms are expanded *w.r.t.* to the signature, to the TRS:

$$\begin{aligned}\mathbb{T}(S_{pz}) = \{\ & \varphi_{pz}(\mathtt{+}(\mathtt{Z},x)) \twoheadrightarrow x,\\ & \varphi_{pz}(\mathtt{Z}) \twoheadrightarrow \bot(\mathtt{Z}),\\ & \varphi_{pz}(\mathtt{S}(y_1)) \twoheadrightarrow \bot(\mathtt{S}(y_1)),\\ & \varphi_{pz}(\mathtt{+}(\mathtt{S}(y_1),y_2)) \twoheadrightarrow \bot(\mathtt{+}(\mathtt{S}(y_1),y_2)),\\ & \varphi_{pz}(\mathtt{+}(\mathtt{+}(y_1,y_2),y_3)) \twoheadrightarrow \bot(\mathtt{+}(\mathtt{+}(y_1,y_2),y_3)),\\ & \varphi_{pz}(\bot(x)) \twoheadrightarrow \bot(x)\ \}\end{aligned}$$

The term $\varphi_{pz}(\mathtt{+}(\mathtt{Z},\mathtt{S}(\mathtt{Z})))$ reduces *w.r.t.* this latter TRS to $\mathtt{S}(\mathtt{Z})$ and $\varphi_{pz}(\mathtt{Z})$ reduces to $\bot(\mathtt{Z})$.

The translation of the sequential application of two strategies (**E4**) includes the translation of the respective strategies and some specific rules. A term $\varphi_{S_1;S_2}(t)$ is reduced by the first rule into a term $\varphi_;(\varphi_{S_2}(\varphi_{S_1}(t)),t)$, which guarantees that the rules of the encoding of S_1 are applied before the ones of S_2. Indeed, a term of the form $\varphi(t)$ can be reduced only if $t \in \mathcal{T}(\mathcal{F})$ or $t = \bot(\_)$ and thus, the rules for φ_{S_2} can be applied to a term $\varphi_{S_2}(\varphi_{S_1}(\_))$ only after $\varphi_{S_1}(\_)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$ (or failure). The original subject t is kept during the evaluation (of $\varphi_;$), so that $\bot(t)$ can be returned if the evaluation of S_1 or S_2 fails (*i.e.* produces a $\bot$) at some point. If $\varphi_{S_2}(\varphi_{S_1}(t))$ evaluates to a term $t' \in \mathcal{T}(\mathcal{F})$, then the evaluation of $\varphi_{S_1;S_2}(t)$ succeeds, and t' is the final result.

In a similar manner, the translation for the choice operator (**E5**) uses a first rule which triggers the application of the rules for S_1. If we start with a term $\varphi_{S_1 \leftrightarrow S_2}(t)$ we first obtain $\varphi_{\leftrightarrow}(\varphi_{S_1}(t))$ and this latter term can be further reduced only after the reduction of $\varphi_{S_1}(t)$ to a term in the signature or to a failure. Recording the origin of the failure and propagating it together with its origin is crucial here. The eventual failure for the reduction of $\varphi_{S_1}(t)$ can be thus detected and the original subject, *i.e.* the term t, can be retrieved and used to trigger the rules for S_2 on $\varphi_{S_2}(t)$.

[2]The rule is in fact expanded into n rewrite rules with n the number of symbols in $\mathcal{F}$.

The translation for a strategy $\mu X \,.\, S$ (**E6**) triggers the application of the rules for S at first, and then each time the symbol φ_X is encountered. As in all the other cases, failure is strictly propagated. There is no rewrite rule for the translation of a strategy variable (**E7**) but we should note that the corresponding φ_X symbol is used when the sub-strategy X is reached during the translation of the strategy S (in $\mu X \,.\, S$).

Example 3.3. The strategy $S_{rpz} = \mu X \,.\, (\texttt{+}(\texttt{Z}, x) \twoheadrightarrow x \;;\; X) \leftarrow\!\!\!+ \mathit{Identity}$ which applies repeatedly (as long as possible) the rewrite rule from Example 3.2 is encoded by:

$$
\begin{array}{rll}
\mathbb{T}(S_{rpz}) = & \{\ \varphi_{rpz}(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_{tpz}(x), & \varphi_{rpz}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\cup & \{\ \varphi_X(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_{tpz}(x), & \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\cup & \{\ \varphi_{tpz}(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!+}(\varphi_{pzX}(x)), & \varphi_{tpz}(\bot(x)) \twoheadrightarrow \bot(x), \\
& \quad \varphi_{\leftarrow\!+}(x \,@\, !\bot(\_)) \twoheadrightarrow x, & \varphi_{\leftarrow\!+}(\bot(x)) \twoheadrightarrow \varphi_{\mathit{Identity}}(x)\ \} \\
\cup & \{\ \varphi_{pzX}(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_{;}(\varphi_X(\varphi_{pz}(x)), x), & \varphi_{pzX}(\bot(x)) \twoheadrightarrow \bot(x), \\
& \quad \varphi_{;}(x \,@\, !\bot(\_), \_) \twoheadrightarrow x, & \varphi_{;}(\bot(\_), x) \twoheadrightarrow \bot(x)\ \} \\
\cup & \mathbb{T}(S_{pz}) \ \cup\ \mathbb{T}(\mathit{Identity}) &
\end{array}
$$

For presentation purposes, we separated the TRS in subsets of rules corresponding to the translation of each operator occurring in the initial strategy. Note that the symbol φ_X used in the rules for the inner sequence can be reduced with the rules generated to handle the recursion operator. The term $\varphi_{rpz}(\texttt{+}(\texttt{Z}, \texttt{+}(\texttt{Z}, \texttt{S}(\texttt{Z}))))$ reduces *w.r.t.* the TRS to $\texttt{S}(\texttt{Z})$.

The rules encoding the traversal operators follow the same principle – the rules corresponding to the translation of the argument strategy S are applied, depending on the traversal operator, to one or all the sub-terms of the subject. For the *All* operator (**E8**), if the application of S to all the sub-terms succeeds (produces terms in $\mathcal{T}(\mathcal{F})$), then the final result is built using the results of each evaluation. If the evaluation of one of the sub-terms produces a $\bot$, a failure with the original subject as origin is returned as a result. Special rules encode the fact that *All* applied to a constant always succeeds; the same behaviour could have been obtained by instantiating the rules for non-constants with $n = 0$, but we preferred an explicit approach for uniformity and efficiency reasons. The strict propagation of failure together with its origin is essential in the case of the *One* operator (**E9**) since it allows for a more economic encoding approach than the duplication of the original subject used for the *All* operator. If the evaluation for one sub-term results in a failure, then the evaluation of the strategy S is triggered on the next one and if S fails on all sub-terms, the final result can be directly built using the origin of these failures which represent nothing else than the original arguments. Note that the failure in case of constants is necessarily encoded by specific rules.

Finally, each binding $X\colon S$ of a context (**E10**) is translated by two rules, including the one that propagates failure. The other rule operates as in the recursive case (**E6**): applying the strategy variable X to a subject t leads to the application of the rules encoding S to t.

Remark 3.4. It is possible to adapt our encoding to a non-deterministic version of $S_1 \leftarrow\!\!\!+ S_2$ and $\mathit{One}(S)$ (cf Remark 2.6). For the choice combinator, the encoding becomes

$$
\begin{array}{rcl}
\mathbb{T}_{nd}(S_1 \leftarrow\!\!\!+ S_2) & = & \mathbb{T}_{nd}(S_1) \ \cup\ \mathbb{T}_{nd}(S_2) \\
& \bigcup & \{\ \varphi_{S_1 \leftarrow\!+ S_2}(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!+}(\varphi_{S_1}(x), \varphi_{S_2}(x)), \quad \varphi_{S_1 \leftarrow\!+ S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
& & \quad \varphi_{\leftarrow\!+}(x \,@\, !\bot(\_), \_) \twoheadrightarrow x, \quad \varphi_{\leftarrow\!+}(\_, x \,@\, !\bot(\_)) \twoheadrightarrow x, \\
& & \quad \varphi_{\leftarrow\!+}(\bot(x), \bot(x)) \twoheadrightarrow \bot(x)\ \}
\end{array}
$$

In this modified encoding, the subject t is duplicated so that the rules of $\mathbb{T}_{nd}(S_1)$ can be applied to $\varphi_{S_1}(t)$ and those of $\mathbb{T}_{nd}(S_2)$ to $\varphi_{S_2}(t)$. If either copy rewrites into a term t' in $\mathcal{T}(\mathcal{F})$, then it can be selected as the final result by the rule $\varphi_{\leftrightarrow}(x \,@\, !\bot(\_), \_) \twoheadrightarrow x$ or its symmetric. If both of them could be eventually rewritten into the terms t'_1, t'_2 in $\mathcal{T}(\mathcal{F})$, the final result is either t'_1 or t'_2 depending on whether $\varphi_{\leftrightarrow}(x \,@\, !\bot(\_), \_) \twoheadrightarrow x$ or $\varphi_{\leftrightarrow}(\_, x \,@\, !\bot(\_)) \twoheadrightarrow x$ is applied first in the corresponding reduction.This non-deterministic choice reflects the non-deterministic nature of the encoded strategy.
The encoding for the non-deterministic *One* follows the same idea.

$$\begin{array}{rcl}
\mathbb{T}_{nd}(\mathit{One}(S)) & = & \mathbb{T}_{nd}(S) \bigcup \{\ \varphi_{\mathit{One}(S)}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \bigcup\limits_{c\in\mathcal{F}^0} \{\ \varphi_{\mathit{One}(S)}(c) \twoheadrightarrow \bot(c)\ \} \\
\bigcup\limits_{f\in\mathcal{F}^+} & \{ & \varphi_{\mathit{One}(S)}(f(x_1,\dots,x_n)) \twoheadrightarrow \varphi_f(\varphi_S(x_1),\dots,\varphi_S(x_n), f(x_1,\dots,x_n)), \\
 & & \varphi_f(\bot(_),\dots,\bot(_), x) \twoheadrightarrow \bot(x), \\
 & & \varphi_f(y \,@\, !\bot(_), _,\dots,_, f(x_1,\dots,x_n)) \twoheadrightarrow f(y, x_2,\dots,x_n), \\
 & & \vdots \\
 & & \varphi_f(_,\dots,_, y \,@\, !\bot(_), f(x_1,\dots,x_n)) \twoheadrightarrow f(x_1,\dots,x_{n-1}, y)\ \}
\end{array}$$

The obtained encoding is similar to the one for *All*: the rules corresponding to the translation S are triggered on all the sub-terms of the subject. We remember the original subject so it can be returned as origin of an eventual failure, or to construct the final result with one of the last n rules. Again, if the encoding of S succeeds on several sub-terms, different results can be obtained depending on which of the last n rules is being applied.

3.2. **Properties of the translation.** The goal of the translation is twofold: use well-established methods and tools for plain TRS in order to prove properties of strategy controlled rewrite rules, and offer a generic compiler for user defined strategies. For both items, it is crucial to have a sound and complete translation, and this turns out to be true in our case.

Theorem 3.5 (Simulation)**.** *Given a term $t \in \mathcal{T}(\mathcal{F})$, a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$,*
(1) $\Gamma \vdash S \circ t \Longrightarrow t'$ *iff* $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow t'$, $t' \in \mathcal{T}(\mathcal{F})$
(2) $\Gamma \vdash S \circ t \Longrightarrow \texttt{Fail}$ *iff* $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$

Sketch. The completeness is shown by induction on the height of the derivation tree and the soundness by induction on the length of the reduction. The base cases consisting of the strategies with a constant length reduction – *Identity*, *Fail*, and the rewrite rule – are straightforward to prove since, in particular, the translation of a rule explicitly encodes matching success and failure. Induction is applied for all the other cases and the corresponding proofs rely on some auxiliary properties.

First, the failure is strictly propagated: if $\mathbb{B}(\Gamma) \cup \mathbb{T}(S) \bullet \varphi_S(\bot(t)) \longrightarrow\!\!\!\!\!\rightarrow u$, then $u = \bot(t)$. This is essential, in particular, for the sequence case where a failure of the first strategy should be strictly propagated as the final result of the overall sequential strategy.

Second, terms in $\mathcal{T}(\mathcal{F})$ are in normal form *w.r.t.* the translation of any strategy and terms of the form $\varphi_S(t)$ are head-rigid *w.r.t.* strategies other than S, *i.e.*, they can be reduced at the head position only by the rules in $\mathbb{T}(S)$ and only if t is a term in the signature. More precisely, if for a strategy S' and a context Γ, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow u$ then $t \in \mathcal{T}(\mathcal{F})$

and $\mathbb{T}(S) \subseteq \mathbb{B}(\Gamma) \cup \mathbb{T}(S')$ (or $S = X$ and $X \in \mathcal{D}om_{\mathcal{X}}(\Gamma)$). This guarantees that the steps in the strategy derivation are encoded accurately by the evaluations *w.r.t.* the rules in the translation.

Finally, the origin of the failure is preserved in the sense that if for a $t \in \mathcal{T}(\mathcal{F})$, $\varphi_S(t)$ reduces to a failure, then the reduct is necessarily $\bot(t)$. This is crucial in particular for the choice strategy: if the (translation of the) first strategy fails, then the (translation of the) second one should be applied on the initial subject. $\square$

The main goal is to prove the termination of some strategy guided system by proving the property for the plain TRS obtained by the translation. As a direct consequence of Theorem 3.5, we obtain that the termination of the TRS encoding a strategy implies the termination of the strategy.

Corollary 3.6 (Termination)**.** *Given a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, S terminates under Γ if $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ is terminating.*

Because of the modular encoding of strategies, the non-termination of the TRS does not necessarily imply the non-termination of the original strategy, and could instead indicate just the non-termination of one of its sub-strategies. Given the shape of the rewrite rules in the encoding, an eventual non-terminating reduction could be always reduced to a non-terminating reduction of a term of the form $\varphi_S(t)$ with φ_S encoding the behaviour of a sub-strategy S and $t \in \mathcal{T}(\mathcal{F})$. For example, the terminating strategies $\mathit{Fail}; \mu X . (\mathit{Identity}; X)$, $\mathit{Identity} \leftarrow\!\!\!+ \mu X . (\mathit{Identity}; X)$, and $a \twoheadrightarrow a; \mu X . (b \twoheadrightarrow b; X)$ use some non-terminating sub-strategies whose encodings are also non-terminating.

Note that even if the non-termination of the encoding reflects the non-termination of the strategy, the counterexample exhibited (by an automatic tool) could concern only a sub-strategy. For example, the strategy $\mathit{All}(\mu X . (\mathit{Identity}; X))$ is non-terminating because of its sub-strategy $\mu X . (\mathit{Identity}; X)$ and an automatic tool would usually provide a counterexample for (the encoding of) this latter strategy.

Another direct consequence of Theorem 3.5 is that the confluence of the strategy is implied by the confluence of the corresponding encoding TRS.

Corollary 3.7 (Confluence)**.** *Given a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, if $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ is confluent then, for any evaluation judgments $\Gamma \vdash S \circ t \Longrightarrow u$ and $\Gamma \vdash S \circ t \Longrightarrow u'$ we have $u = u'$.*

Moreover, the encoding of a deterministic strategy always leads to a confluent TRS:

Lemma 3.8. *Given a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, the TRS $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ is confluent.*

Sketch. We split $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ into two TRS and show that they are both linear, confluent and orthogonal to each other, *i.e.* there is no overlap between a rule from one and a rule from the other. As a consequence, we have the confluence of $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ [32]. $\square$

4. Meta-encoding rewriting strategies with rewrite rules

The strategy translation of Section 3 generates a potentially large number of rewrite rules essentially because of the way *All* and *One* are encoded, and this may impact the efficiency of the corresponding implementation or prevent an automatic tool from deciding termination.

We propose in this section a meta-encoding of terms which allows for a more economic encoding of strategies into rewrite rules.

4.1. **Meta-level representation of terms.** The non-failure $!\bot(\_)$ has been encoded so far by the set of terms $\bigcup_{f\in\mathcal{F}}\{f(x_1,\ldots,x_n)\}$, with $ar(f)=n$, and thus all rule schemas in Figure 2 using this construction are expanded into a number of rules proportional to the number of symbols in the signature. The size of the encodings for *All* and *One* in the translation depends also on the signature $\mathcal{F}$, as the number of rules generated grows with the number of symbols and with their arities. To avoid this potential explosion, we represent terms in $\mathcal{T}(\mathcal{F})$ using a meta-level application and lists: roughly, we write $f(t_1,\ldots,t_n)$ as $\mathtt{appl}(f, args)$, where $args$ is the list of the meta-encoded arguments, and a constant c is represented as $\mathtt{appl}(c,\mathtt{nil})$.

Formally, given a signature $\mathcal{F}$, we define a meta-signature $\mathcal{F}_{\mathtt{appl}}=\{\mathtt{appl}, ::, \mathtt{nil}\}\cup\{\ulcorner f\urcorner \mid f\in\mathcal{F}\}$ so that $\mathtt{appl}$ and $::$ are of arity 2, and $\mathtt{nil}$ and $\ulcorner f\urcorner$ are of arity 0 for all $f\in\mathcal{F}$. The symbols $::$ and $\mathtt{nil}$ are the usual building blocks for lists, and we use $::$ as a right-associative infix operator, so that $x_1 :: (x_2 :: \mathtt{nil})$ is written $x_1 :: x_2 :: \mathtt{nil}$. Given a term $t\in\mathcal{T}(\mathcal{F},\mathcal{X})$, we define its meta-level encoding $\ulcorner t\urcorner\in\mathcal{T}(\mathcal{F}_{\mathtt{appl}},\mathcal{X})$ as $\ulcorner f(t_1,\ldots,t_n)\urcorner = \mathtt{appl}(\ulcorner f\urcorner, \ulcorner t_1\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \mathtt{nil})$, $\ulcorner c\urcorner = \mathtt{appl}(\ulcorner c\urcorner, \mathtt{nil})$ if $c\in\mathcal{F}^0$, and $\ulcorner x\urcorner = x$ if $x\in\mathcal{X}$.

4.2. **Strategy meta-encoding.** Given the above meta-encoding of terms, we define a translation $\mathbb{T}_{\mathcal{M}}$ of rewriting strategies on terms in $\mathcal{T}(\mathcal{F})$ into plain rewrite rules on terms built from $\mathcal{F}_{\mathtt{appl}}$ extended with a set of generated φ symbols, with the symbols $\bot$, $\bot_{list}$ encoding failure, and with the symbols for lists manipulation presented in Figure 4. The translations of the elementary strategies and of the control combinators are almost the same as in Section 3, so we present in Figure 3 only the strategies whose encoding differs the most from Figure 2, and we give the complete translation in Appendix 9 (Figure 6).

First, note that meta-encoded terms are of the form $\mathtt{appl}(\_,\_)$, so instead of using $x \,@\, !\bot(\_)$ (as in the rule schemas in Figure 2) to filter all the ground terms of the original signature, we directly use $x \,@\, \mathtt{appl}(\_,\_)$. As explained in the previous section, the rule schemas relying on $x \,@\, !\bot(\_)$ are expanded into a set of rewrite rules whose cardinality depends on the original signature. By using the term $x \,@\, \mathtt{appl}(\_,\_)$ instead of this anti-term, each set of rules represented by a rule schema in the original encoding is replaced by only one rewrite rule in the meta-encoding. Consequently, the number of rules in the meta-encoding can be significantly smaller than in the previous encoding.

We should point out that the meta-encoding of a rewrite rule (**ME3**) uses $\ulcorner !l\urcorner$ and not $!\ulcorner l\urcorner$, meaning that we meta-encode the terms of $\mathcal{T}(\mathcal{F})$ that do not match l, instead of considering all the terms of $\mathcal{T}(\mathcal{F}_{\mathtt{appl}})$ that do not match $\ulcorner l\urcorner$. Indeed, $\mathcal{F}_{\mathtt{appl}}$ is more expressive than $\mathcal{F}$ since we can write terms $\mathtt{appl}(\ulcorner f\urcorner, args)$ where the size of $args$ differs from the arity of f in $\mathcal{F}$. Such ill-formed terms are matched by $!\ulcorner l\urcorner$, but not by $\ulcorner !l\urcorner$; however, they do not have to be considered since they are never produced during the evaluation of a meta-encoded term *w.r.t.* a strategy meta-encoding (see Lemma 4.3). As a result, we generate as many rules to represent this anti-pattern with the meta-encoding as with the original encoding.

Example 4.1. If we consider the signature in Example 2.1, the meta-encoding of the strategy $S_{pz} = \mathtt{+}(\mathtt{Z},x) \twoheadrightarrow x$ contains the same rules as in the regular encoding in Example 3.2, except

$$
\begin{array}{llll}
(\mathbf{ME3}) & \mathbb{T}_{\mathcal{M}}(l \twoheadrightarrow r) & = & \{\ \varphi_{l\twoheadrightarrow r}(\ulcorner l \urcorner) \twoheadrightarrow \ulcorner r \urcorner,\quad \varphi_{l\twoheadrightarrow r}(x \mathbin{@} \ulcorner !l \urcorner) \twoheadrightarrow \bot(x), \\
& & & \ \ \varphi_{l\twoheadrightarrow r}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{ME8}) & \mathbb{T}_{\mathcal{M}}(All(S)) & = & \mathbb{T}_{\mathcal{M}}(S)\ \cup\ \mathbb{L} \\
& & \bigcup & \{\ \varphi_{All(S)}(\bot(x)) \twoheadrightarrow \bot(x), \\
& & & \ \ \varphi_{All(S)}(\mathtt{appl}(f, args)) \twoheadrightarrow \mathtt{propag}(\mathtt{appl}(f, \varphi^{list}_{All(S)}(args))), \\
& & & \ \ \varphi^{list}_{All(S)}(h :: q) \twoheadrightarrow \varphi'_{All(S)}(\varphi_S(h), q, h :: \mathtt{nil}, \mathtt{nil}), \\
& & & \ \ \varphi^{list}_{All(S)}(\mathtt{nil}) \twoheadrightarrow \mathtt{nil}, \\
& & & \ \ \varphi'_{All(S)}(\bot(\_), todo, r\_tried, \_) \twoheadrightarrow \bot_{list}(\mathtt{rconcat}(r\_tried, todo)), \\
& & & \ \ \varphi'_{All(S)}(x \mathbin{@} \mathtt{appl}(\_,\_), \mathtt{nil}, \_, r\_done) \twoheadrightarrow \mathtt{rev}(x :: r\_done), \\
& & & \ \ \varphi'_{All(S)}(x \mathbin{@} \mathtt{appl}(\_,\_), h :: q, r\_tried, r\_done) \twoheadrightarrow \\
& & & \qquad\qquad \varphi'_{All(S)}(\varphi_S(h), q, h :: r\_tried, x :: r\_done)\ \} \\
(\mathbf{ME9}) & \mathbb{T}_{\mathcal{M}}(One(S)) & = & \mathbb{T}_{\mathcal{M}}(S)\ \cup\ \mathbb{L} \\
& & \bigcup & \{\ \varphi_{One(S)}(\bot(x)) \twoheadrightarrow \bot(x), \\
& & & \ \ \varphi_{One(S)}(\mathtt{appl}(f, args)) \twoheadrightarrow \mathtt{propag}(\mathtt{appl}(f, \varphi^{list}_{One(S)}(args))), \\
& & & \ \ \varphi^{list}_{One(S)}(h :: q) \twoheadrightarrow \varphi'_{One(S)}(\varphi_S(h), q, h :: \mathtt{nil}), \\
& & & \ \ \varphi^{list}_{One(S)}(\mathtt{nil}) \twoheadrightarrow \bot_{list}(\mathtt{nil}), \\
& & & \ \ \varphi'_{One(S)}(\bot(\_), \mathtt{nil}, r\_tried) \twoheadrightarrow \bot_{list}(\mathtt{rev}(r\_tried)), \\
& & & \ \ \varphi'_{One(S)}(\bot(\_), h :: q, r\_tried) \twoheadrightarrow \varphi'_{One(S)}(\varphi_S(h), q, h :: r\_tried), \\
& & & \ \ \varphi'_{One(S)}(x \mathbin{@} \mathtt{appl}(\_,\_), todo, \_ :: r\_tried) \twoheadrightarrow \\
& & & \qquad\qquad \mathtt{rconcat}(r\_tried, x :: todo)\quad \}
\end{array}
$$

Figure 3: Strategy translation for meta-encoded terms; $\mathbb{L}$ is defined in Figure 4.

that we replace the terms of the original signature by their meta-level encodings:[3]

$$
\begin{array}{ll}
\mathbb{T}_{\mathcal{M}}(S_{pz}) = \{ & \varphi_{pz}(\mathtt{appl}(\mathtt{+}, \mathtt{appl}(\mathtt{Z}, \mathtt{nil}) :: x :: \mathtt{nil})) \twoheadrightarrow x, \\
& \varphi_{pz}(\mathtt{appl}(\mathtt{Z}, \mathtt{nil})) \twoheadrightarrow \bot(\mathtt{appl}(\mathtt{Z}, \mathtt{nil})), \\
& \varphi_{pz}(\mathtt{appl}(\mathtt{S}, y_1 :: \mathtt{nil})) \twoheadrightarrow \bot(\mathtt{appl}(\mathtt{S}, y_1 :: \mathtt{nil})), \\
& \varphi_{pz}(\mathtt{appl}(\mathtt{+}, \mathtt{appl}(\mathtt{S}, y_1 :: \mathtt{nil}) :: y_2 :: \mathtt{nil})) \twoheadrightarrow \\
& \qquad\qquad \bot(\mathtt{appl}(\mathtt{+}, \mathtt{appl}(\mathtt{S}, y_1 :: \mathtt{nil}) :: y_2 :: \mathtt{nil})), \\
& \varphi_{pz}(\mathtt{appl}(\mathtt{+}, \mathtt{appl}(\mathtt{+}, y_1 :: y_2 :: \mathtt{nil}) :: y_3 :: \mathtt{nil})) \twoheadrightarrow \\
& \qquad \bot(\mathtt{appl}(\mathtt{+}, \mathtt{appl}(\mathtt{+}, y_1 :: y_2 :: \mathtt{nil}) :: y_3 :: \mathtt{nil})), \\
& \varphi_{pz}(\bot(x)) \twoheadrightarrow \bot(x)\ \}
\end{array}
$$

The following meta-encoding is obtained for the *Identity* strategy:

$$
\mathbb{T}_{\mathcal{M}}(Identity) = \{\ \varphi_{Identity}(\mathtt{appl}(y_1, y_2)) \twoheadrightarrow \mathtt{appl}(y_1, y_2),\ \ \varphi_{Identity}(\bot(x)) \twoheadrightarrow \bot(x)\ \}
$$

[3]For readability reasons we write in what follows $\mathtt{appl}(f, \ldots)$ instead of $\mathtt{appl}(\ulcorner f \urcorner, \ldots)$ for all $f \in \mathcal{F}$.

$$
\begin{array}{rlll}
\mathbb{L} & = \{ & \mathtt{append}(\mathtt{nil}, x) & \twoheadrightarrow x :: \mathtt{nil}, \\
& & \mathtt{append}(h :: q, x) & \twoheadrightarrow h :: \mathtt{append}(q, x), \\
& & \mathtt{rev}(\mathtt{nil}) & \twoheadrightarrow \mathtt{nil}, \\
& & \mathtt{rev}(h :: q) & \twoheadrightarrow \mathtt{append}(\mathtt{rev}(q), h), \\
& & \mathtt{rconcat}(\mathtt{nil}, x) & \twoheadrightarrow x, \\
& & \mathtt{rconcat}(h :: q, x) & \twoheadrightarrow \mathtt{rconcat}(q, h :: x), \\
& & \mathtt{propag}(\mathtt{appl}(f, \bot_{list}(args))) & \twoheadrightarrow \bot(\mathtt{appl}(f, args)), \\
& & \mathtt{propag}(\mathtt{appl}(f, h :: q)) & \twoheadrightarrow \mathtt{appl}(f, h :: q), \\
& & \mathtt{propag}(\mathtt{appl}(f, \mathtt{nil})) & \twoheadrightarrow \mathtt{appl}(f, \mathtt{nil}) \}
\end{array}
$$

Figure 4: Auxiliary symbols and rewrite rules for lists manipulation.

and the strategy $S_{rpz} = \mu X\,.\,(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \;;\; X) \leftarrow\!\!\!\!+ \mathit{Identity}$ from Example 3.3 is encoded by:

$$
\begin{array}{rll}
\mathbb{T}_{\mathcal{M}}(S_{rpz}) = & \{ \varphi_{rpz}(x \,@\, \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_{tpz}(x), & \varphi_{rpz}(\bot(x)) \twoheadrightarrow \bot(x) \ \} \\
\cup & \{ \varphi_{X}(x \,@\, \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_{tpz}(x), & \varphi_{X}(\bot(x)) \twoheadrightarrow \bot(x) \ \} \\
\cup & \{ \varphi_{tpz}(x \,@\, \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_{\leftarrow\!+}(\varphi_{pzX}(x)), & \varphi_{tpz}(\bot(x)) \twoheadrightarrow \bot(x), \\
& \quad \varphi_{\leftarrow\!+}(x \,@\, \mathtt{appl}(\_,\_)) \twoheadrightarrow x, & \varphi_{\leftarrow\!+}(\bot(x)) \twoheadrightarrow \varphi_{Identity}(x) \ \} \\
\cup & \{ \varphi_{pzX}(x \,@\, \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_{;}(\varphi_{X}(\varphi_{pz}(x)), x), & \varphi_{pzX}(\bot(x)) \twoheadrightarrow \bot(x), \\
& \quad \varphi_{;}(x \,@\, \mathtt{appl}(\_,\_), \_) \twoheadrightarrow x, & \varphi_{;}(\bot(\_), x) \twoheadrightarrow \bot(x) \ \} \\
\cup & \mathbb{T}_{\mathcal{M}}(S_{pz}) \cup \mathbb{T}_{\mathcal{M}}(\mathit{Identity}) &
\end{array}
$$

In contrast with the translations of the previous section, only the encoding of the strategy S_{pz} depends on the considered signature, all the other rules in the encoding would be the same independently of the signature.

The meta-level representation of the terms has a more important impact on the way $\mathit{All}(S)$ and $\mathit{One}(S)$ are translated. When applying (the encoding of) one of these strategies to a term $\mathtt{appl}(f, args)$ we want to apply S to each of the elements of the list $args$ to check if S succeeds or not on these arguments. To do so, we have to manipulate explicitly lists of arguments and we introduce thus some extra symbols and the corresponding rewrite rules to handle them: $\mathtt{append}(list, x)$ adds the element x at the end of $list$, $\mathtt{rev}(list)$ computes the reverse of $list$, $\mathtt{rconcat}(list_1, list_2)$ concatenates the reverse of $list_1$ with $list_2$, and $\mathtt{propag}$ propagates the failure or the success of the application of a strategy on the arguments of a meta-level term to the whole term. The rewrite rules for these symbols are given in Figure 4.

Given a subject $\mathtt{appl}(f, args)$, the encoding of $\mathit{All}(S)$ first checks whether the list $args$ is empty or not with the rules for the symbol $\varphi^{list}_{All(S)}$. If it is empty, then $\mathit{All}(S)$ has been applied to a (meta-)constant $\mathtt{appl}(f, \mathtt{nil})$, and the result should be the term itself, which is the case because $\varphi^{list}_{All(S)}(\mathtt{nil}) \twoheadrightarrow \mathtt{nil}$ and $\mathtt{propag}(\mathtt{appl}(f, \mathtt{nil})) \twoheadrightarrow \mathtt{appl}(f, \mathtt{nil})$. If $args = t_1 :: t_2 :: \ldots :: t_n :: \mathtt{nil}$, we go through the list of arguments, using terms of the form $\varphi'_{All(S)}(x, todo, r\_tried, r\_done)$, where x is the element being reduced, initialized successively with $\varphi_S(t_i)$ for $1 \le i \le n$, $todo$ is a list containing the terms $t_{i+1} \ldots t_n$ remaining to reduce, r_tried is a list containing the terms $t_i \ldots t_1$ already reduced, and r_done is the list containing the reducts $t'_i \ldots t'_1$ of the terms in r_tried. If $\varphi_S(t_i)$ reduces to $\bot(t_i)$, then we abort the whole process by producing first $\bot_{list}(\mathtt{rconcat}(r\_tried, todo))$, which reduces to $\bot_{list}(args)$, and then propagating failure with $\mathtt{propag}$ to generate the term $\bot(\mathtt{appl}(f, args))$. If reducing $\varphi_S(t_i)$ produces a meta-term t'_i, then we store this result in r_done and proceed with the next term in $todo$. If there are no terms left to be evaluated in $todo$, we build the final result using the last result t'_n and the previously reduced arguments $t'_{n-1} \dots t'_1$ to obtain $\mathtt{appl}(f, t'_1 :: \dots :: t'_n :: \mathtt{nil})$.

Example 4.2. The meta-encoding of the strategy $S_{all} = All(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x)$ consists of:

$$\begin{array}{rl} \mathbb{T}_{\mathcal{M}}(S_{all}) = \{ & \varphi_{all}(\bot(x)) \twoheadrightarrow \bot(x), \\ & \varphi_{all}(\mathtt{appl}(f, args)) \twoheadrightarrow \mathtt{propag}(\mathtt{appl}(f, \varphi^{list}_{all}(args))), \\ & \varphi^{list}_{all}(h :: q) \twoheadrightarrow \varphi'_{all}(\varphi_{pz}(h), q, h :: \mathtt{nil}, \mathtt{nil}), \\ & \varphi^{list}_{all}(\mathtt{nil}) \twoheadrightarrow \mathtt{nil}, \\ & \varphi'_{all}(\bot(\_), td, rt, \_) \twoheadrightarrow \bot_{list}(\mathtt{rconcat}(rt, td)), \\ & \varphi'_{all}(\mathtt{appl}(f, args), \mathtt{nil}, \_, rd) \twoheadrightarrow \mathtt{rev}(\mathtt{appl}(f, args) :: rd), \\ & \varphi'_{all}(\mathtt{appl}(f, args), h :: q, rt, rd) \twoheadrightarrow \\ & \qquad\qquad \varphi'_{all}(\varphi_{pz}(h), q, h :: rt, \mathtt{appl}(f, args) :: rd) \ \} \\ \cup & \mathbb{T}_{\mathcal{M}}(S_{pz}) \ \cup \ \mathbb{L} \end{array}$$

The translation $\mathbb{T}(All(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x))$ would contain much more rules and their number depends on the considered signature. We can see in Example 5.4 the impact of the signature on the size of the encoding using the original terms instead of their meta-level representation.

The encoding of $One(S)$ follows the same principles. If $args = t_1 :: t_2 :: \dots :: t_n :: \mathtt{nil}$, then we explore the list with $\varphi'_{One(S)}(x, todo, r\_tried)$, where x is initialized successively with $\varphi_S(t_i)$ until the encoding of S succeeds on one of these terms. The list $todo$ contains what is left to try (*i.e.* $t_{i+1} :: \dots :: t_n :: \mathtt{nil}$), and r_tried contains $t_i :: t_{i-1} :: \dots :: t_1 :: \mathtt{nil}$. If x becomes a meta-term t'_i, then we rewrite it into $\mathtt{rconcat}(t_{i-1} :: \dots :: t_1 :: \mathtt{nil}, t'_i :: t_{i+1} :: \dots :: t_n :: \mathtt{nil})$ which then reduces to the required result. If S fails on t_i (*i.e.* x becomes $\bot(t_i)$) and $todo$ is not empty, then we consider the next term by reducing to $\varphi'_{One(S)}(\varphi_S(t_{i+1}), todo, t_{i+1} :: r\_tried)$. Otherwise, we reduce to $\bot_{list}(\mathtt{rev}(r\_tried))$, which produces $\bot_{list}(args)$, and the rules for $\mathtt{propag}$ then generate $\bot(\mathtt{appl}(f, args))$, as wished.

4.3. **Properties of the meta-encoding.** We state now the correctness of the meta-encoding by first establishing the correspondence between $\mathbb{T}_{\mathcal{M}}$ and the translation of Section 3. The next lemma also shows that applying the strategy meta-encoding to a meta-encoded term results in a meta-encoded term, as expected. We extend $\ulcorner \cdot \urcorner$ to $\bot$ by defining $\ulcorner \bot(t) \urcorner = \bot(\ulcorner t \urcorner)$.

Lemma 4.3. *Given a term $t \in \mathcal{T}(\mathcal{F})$, a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$,*

(1) *if $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow u$ with $u \in \mathcal{T}(\mathcal{F})$ or $u = \bot(t)$, then $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner u \urcorner$;*
(2) *if $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow t''$ and $t'' \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$, then there exists t' such that $\ulcorner t' \urcorner = t''$ and $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow t'$.*
(3) *if $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(t'')$ and $t'' \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$, then $t'' = \ulcorner t \urcorner$ and $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$.*

Sketch. The proof is by induction on S. The correspondence is direct for the reduction steps which do not involve All or One. In these two cases, we relate how $\mathbb{T}_{\mathcal{M}}(All(S))$ or $\mathbb{T}_{\mathcal{M}}(One(S))$ behaves on $\mathtt{appl}(\ulcorner f \urcorner, \ulcorner t_1 \urcorner :: \dots :: \ulcorner t_n \urcorner :: \mathtt{nil})$ depending on how $\mathbb{T}_{\mathcal{M}}(S)$ behaves on each of the $\ulcorner t_i \urcorner$. For example, we show that $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t_i' \urcorner$ with $\ulcorner t_i' \urcorner \neq \bot(\ulcorner t_i \urcorner)$ for all i iff

$$\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi'_{All(S)}(\varphi_S(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_1 \urcorner :: \mathtt{nil}, \mathtt{nil}) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t_1' \urcorner :: \ldots :: \ulcorner t_n' \urcorner :: \mathtt{nil}$$

From there the induction hypothesis tells us that $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t_i' \urcorner$ iff $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t_i) \longrightarrow\!\!\!\!\!\rightarrow t_i'$, and then we can show that $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t_i) \longrightarrow\!\!\!\!\!\rightarrow t_i'$ iff $\mathbb{T}(All(S)) \cup \mathbb{B}(\Gamma) \bullet f(t_1, \ldots, t_n) \longrightarrow\!\!\!\!\!\rightarrow f(t_1', \ldots, t_n')$ to conclude. In total, we have four different cases (depending on whether the applications of $All(S)$ or $One(S)$ succeed or not), which are treated similarly. □

Combined with Theorem 3.5 this result allows us to deduce the correspondence between the strategy application and $\mathbb{T}_{\mathcal{M}}$.

Theorem 4.4 (Simulation)**.** *Given a term $t \in \mathcal{T}(\mathcal{F})$, a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$,*
(1) $\Gamma \vdash S \circ t \Longrightarrow t'$ *iff* $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t' \urcorner$, $t' \in \mathcal{T}(\mathcal{F})$
(2) $\Gamma \vdash S \circ t \Longrightarrow \mathtt{Fail}$ *iff* $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(\ulcorner t \urcorner)$

Proof. By Lemma 4.3 and Theorem 3.5. □

5. Encoding rewriting strategies with typed rewrite rules

Many programming languages use type systems to classify values and expressions into types, to define how those types can be manipulated and how they interact. In logic [29], many-sorted signatures are used similarly to partition the universe into disjoint subsets, one for every sort, and a many-sorted logic naturally leads to a type theory [23]. Indeed, sorts of many-sorted signatures are also known as algebraic data-types in programming languages.

When using many-sorted signatures the strategy translations should guarantee that all the generated terms and rewrite rules respect the corresponding construction constraints. Traversal strategies propagate the application of a given strategy to (sub-)terms of potentially different sorts and consequently the strategies we consider here are intrinsically polymorphic. This polymorphic nature should be clearly retrieved in the encoding and thus, the sorts of the symbols generated during the translation should cope with this constraint. To support this behaviour we consider many-sorted signatures with potentially overloaded symbols and we show that the proposed translations can be adapted easily to work even when overloaded symbols are not supported by the targeted language.

We consider in what follows sort preserving rewrite rules and consequently sort preserving strategies. We propose a translation generating sort preserving TRS which are, as before, faithful encodings of the corresponding strategies.

5.1. **Many-sorted signatures and term rewriting systems.** A *many-sorted signature* $\Sigma = (\mathcal{S}, \mathcal{F})$, or simply Σ, consists of a set of sorts $\mathcal{S}$ and a set of symbols $\mathcal{F}$. A symbol f with *domain* $w = s_1 \times \ldots \times s_n \in \mathcal{S}^*$ and *co-domain* s is written $f : w \mapsto s$; n is its arity and $w \mapsto s$ its profile. Symbols can be overloaded, *i.e.* a symbol f can have profiles $w \mapsto s$ and $w' \mapsto s'$ with $w \neq w'$.

We write $\mathcal{F}_s$ for the subset of symbols of codomain s. Variables are also sorted and $x : s$ means that variable x has sort s. We sometimes annotate the name of a variable by its sort and use, for example, x^s to implicitly indicate a variable of sort s. The set $\mathcal{X}_s$ denotes a set of variables of sort s and $\mathcal{X} = \bigcup_{s\in\mathcal{S}} \mathcal{X}_s$ is the set of sorted variables.

The set of terms of sort s, denoted $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$ is the smallest set containing $\mathcal{X}_s$ and such that $f(t_1, \ldots, t_n)$ is in $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$ whenever $f : s_1\times \ldots \times s_n \mapsto s$ and $t_i \in \mathcal{T}_{s_i}(\mathcal{F}, \mathcal{X})$ for $i \in [1, n]$. We write $t : s$ when the term t is of sort s, *i.e.* when $t \in \mathcal{T}_s(\mathcal{F}, \mathcal{X})$. The set of *sorted terms* is defined as $\mathcal{T}_\mathcal{S}(\mathcal{F}, \mathcal{X}) = \bigcup_{s\in\mathcal{S}} \mathcal{T}_s(\mathcal{F}, \mathcal{X})$. Sorted substitutions are defined as mappings σ from sorted variables to sorted terms such that if $x : s$ then $\sigma(x) \in \mathcal{T}_s(\mathcal{F}, \mathcal{X})$. Note that for any such sorted substitution σ, $t : s$ iff $\sigma(t) : s$.

A *sorted rewrite rule* is a rewrite rule $l \twoheadrightarrow r$ with $l, r : s$ and a sorted TRS is a set of sorted rewrite rules inducing a corresponding rewriting relation over sorted terms. Given two sorted terms $l, t : s$, we say that l matches t if there exists a sorted substitution σ such that $t = \sigma(l)$; in this case we say that t is a sorted instance of l.

5.2. **Typed encoding of rewriting strategies.** Given a strategy built over a many-sorted signature, the translation defined in Section 3 can still be used to generate a faithful encoding in the sense of Theorem 3.5. Since any term in $\mathcal{T}_\mathcal{S}(\mathcal{F})$ is also a term in $\mathcal{T}(\mathcal{F})$, the termination of the (unsorted) TRS encoding the strategy guarantees the termination of the sorted strategy. Nevertheless, we can use the extra information provided by sorts to refine this translation and remove the generated rewrite rules that are useless, because they cannot be applied to sorted terms. More importantly, this translation cannot be used as a strategy compiler if the target language is many-sorted and accepts only sorted terms and rewrite rules.

The presence of ill-sorted terms in the encoding generated by the translation in Figure 2 is essentially due to the unsorted semantics we have considered so far for anti-terms. In an unsorted world, the anti-term $!t$ represents all the terms which do not match t, and $!\bot(x)$ denotes all the ground terms in $\mathcal{T}(\mathcal{F})$. We adapt these notions to a many-sorted setting as follows: given a term t of sort s, we write $!^s t$ for the terms of sort s which do not match t, and given a sort s we write $!^s\bot(x)$ to denote all the ground sorted terms in $\mathcal{T}_s(\mathcal{F})$. Note that $\bigcup_{s\in\mathcal{S}}\{!^s\bot(x)\}$ denotes all the ground *sorted* terms $\mathcal{T}_\mathcal{S}(\mathcal{F})$ of the original signature.

Example 5.1. Consider the many-sorted signature $(\mathcal{S}, \mathcal{F})$ where $\mathcal{S} = \{\texttt{Nat}, \texttt{Bool}\}$ and $\mathcal{F} = \mathcal{F}_{\texttt{Nat}} \cup \mathcal{F}_{\texttt{Bool}}$ with $\mathcal{F}_{\texttt{Nat}} = \{\texttt{Z} : \mapsto \texttt{Nat},\ \texttt{S} : \texttt{Nat} \mapsto \texttt{Nat},\ \texttt{+} : \texttt{Nat}\times\texttt{Nat} \mapsto \texttt{Nat}\}$, $\mathcal{F}_{\texttt{Bool}} = \{\texttt{true} : \mapsto \texttt{Bool},\ \texttt{false} : \mapsto \texttt{Bool},\ \texttt{odd} : \texttt{Nat} \mapsto \texttt{Bool},\ \texttt{even} : \texttt{Nat} \mapsto \texttt{Bool}\}$. The anti-term $!^{\texttt{Nat}}\texttt{+}(\texttt{Z}, x)$ denotes exactly the sorted terms in $\mathcal{T}_{\texttt{Nat}}(\mathcal{F})$ matched by $\texttt{Z}$, $\texttt{S}(x_1)$, $\texttt{+}(\texttt{S}(x_1), x_2)$ or $\texttt{+}(\texttt{+}(x_1, x_2), x_3)$. In particular, it does not denote the (ill-sorted) term $\texttt{+}(\texttt{true}, \texttt{Z})$. The anti-term $!^{\texttt{Bool}}\texttt{+}(\texttt{Z}, x)$ denotes the sorted terms in $\mathcal{T}_{\texttt{Bool}}(\mathcal{F})$ matched by $\texttt{true}$, $\texttt{false}$, $\texttt{odd}(x_1)$ or $\texttt{even}(x_1)$.

The anti-term $!^{\texttt{Bool}}\bot(x)$ denotes all sorted instances of $\texttt{true}$, $\texttt{false}$, $\texttt{odd}(x_1)$ and $\texttt{even}(x_1)$ while $!^{\texttt{Nat}}\bot(x)$ denotes all sorted instances of $\texttt{Z}$, $\texttt{S}(x_1)$ and $\texttt{+}(x_1, x_2)$. The union of all these instances represent indeed all the sorted terms in $t \in \mathcal{T}_\mathcal{S}(\mathcal{F})$.

We now adapt the unsorted translation defined in Section 3 to accommodate many-sorted signatures. Given a sorted signature $(\mathcal{S}, \mathcal{F})$, we define the signature $(\mathcal{S}, \overline{\mathcal{F}})$ where the sorts are unchanged, and $\mathcal{F}$ is extended with the generated φ symbols and $\bot$. Unlike in the unsorted case, we have to specify their profile: the symbols $\bot$ and $\varphi_{\mathit{Identity}}$, φ_{Fail},

$\varphi_{l\twoheadrightarrow r}$, $\varphi_{S_1;S_2}$, $\varphi_{S_1\leftarrow\!\!\!+S_2}$, $\varphi_{\leftarrow\!\!\!+}$, $\varphi_{\mu X.S}$, φ_X, $\varphi_{All(S)}$, $\varphi_{One(S)}$ have sort $s\mapsto s$ for all $s\in\mathcal{S}$, and $\varphi_; : s\times s\mapsto s$ for all $s\in\mathcal{S}$. These symbols are overloaded, since they have as many profiles as there are sorts in $\mathcal{S}$. In contrast, the profiles of the generated symbols of the form φ_f and φ_{f_i} depend on the profile of f: if $f : s_1\times\ldots\times s_n\mapsto s\in\mathcal{F}_s^+$ then $\varphi_f : s_1\times\ldots\times s_n\times s\mapsto s$ and $\varphi_{f_i} : s_1\times\ldots\times s_n\mapsto s$.

The translation $\mathbb{T}_{\mathcal{S}}(S)$ transforms the strategy S into a set of sorted rewrite rules; since its definition is the same as for the unsorted case (Figure 2), except for extra sort annotations on variables and $!$, we give the rules only in Appendix 10 (Figure 7). Even though the definitions of $\mathbb{T}$ and $\mathbb{T}_{\mathcal{S}}$ are almost the same, taking sorts into account could have an important impact on the number of rules generated in the resulting encoding. Sorts may introduce duplication: since $\bot$ has now a profile, for every rule of the form $\varphi(\bot(x))\twoheadrightarrow\bot(x)$ generated in the unsorted case, we generate now the set of rules $\bigcup_{s\in\mathcal{S}}\{\varphi(\bot(x^s))\twoheadrightarrow\bot(x^s)\}$. In contrast, expanding $!\bot(\_)$ in the unsorted case produces as many rules as considering $\bigcup_{s\in\mathcal{S}}\{!^s\bot(x)\}$ in the sorted case, assuming $\mathcal{F}$ has as many elements as $\bigcup_{s\in\mathcal{S}}\mathcal{F}_s$. As detailed in Section 6.4, $!\bot(\_)$ is expanded into the set of terms $\bigcup_{f\in\mathcal{F}}\{f(x_1,\ldots,x_{ar(f)})\}$, while $\bigcup_{s\in\mathcal{S}}\{!^s\bot(x)\}$ becomes the generally smaller set $\bigcup_{s\in\mathcal{S}}\bigcup_{f\in\mathcal{F}_s}\{f(x_1,\ldots,x_n)\}$.

Example 5.2. If we consider the many-sorted signature in Example 5.1, we obtain the following encoding for the *Identity* strategy when expanding the rule schemas in Figure 7:

$$\begin{array}{lll}\mathbb{T}_{\mathcal{S}}(\mathit{Identity})=\{ & \varphi_{\mathit{Identity}}(\mathtt{Z})\twoheadrightarrow\mathtt{Z} & \varphi_{\mathit{Identity}}(\mathtt{true})\twoheadrightarrow\mathtt{true},\\ & \varphi_{\mathit{Identity}}(\mathtt{S}(y_1))\twoheadrightarrow\mathtt{S}(y_1), & \varphi_{\mathit{Identity}}(\mathtt{false})\twoheadrightarrow\mathtt{false},\\ & \varphi_{\mathit{Identity}}(\mathtt{+}(y_1,y_2))\twoheadrightarrow\mathtt{+}(y_1,y_2), & \varphi_{\mathit{Identity}}(\mathtt{odd}(y_1))\twoheadrightarrow\mathtt{odd}(y_1),\\ & \varphi_{\mathit{Identity}}(\bot(x))\twoheadrightarrow\bot(x), & \varphi_{\mathit{Identity}}(\mathtt{even}(y_1))\twoheadrightarrow\mathtt{even}(y_1),\\ & & \varphi_{\mathit{Identity}}(\bot(z))\twoheadrightarrow\bot(z)\ \}\end{array}$$

To improve readability, we do not annotate variables but instead use an environment mapping variables to sorts: here, we have $x,y_1,y_2:\mathtt{Nat}$ and $z:\mathtt{Bool}$. Besides, the signature is enriched in this case with the symbols $\{\ \bot:\mathtt{Nat}\mapsto\mathtt{Nat},\ \ \varphi_{\mathit{Identity}}:\mathtt{Nat}\mapsto\mathtt{Nat},\ \ \bot:\mathtt{Bool}\mapsto\mathtt{Bool},\ \varphi_{\mathit{Identity}}:\mathtt{Bool}\mapsto\mathtt{Bool}\ \}$.

For this signature, the unsorted encoding would contain the same rewrite rules modulo syntactic equivalence, *i.e.* the rules in $\mathbb{T}(\mathit{Identity})$ are those in $\mathbb{T}_{\mathcal{S}}(\mathit{Identity})$ with the rules $\varphi_{\mathit{Identity}}(\bot(x))\twoheadrightarrow\bot(x)$ and $\varphi_{\mathit{Identity}}(\bot(z))\twoheadrightarrow\bot(z)$ collapsed into only one rule.

The unsorted and sorted encodings differ more on how they handle rewrite rules and traversal combinators. As explained before, the anti-pattern $\varphi_{l\twoheadrightarrow r}(x@!l)\twoheadrightarrow\bot(x)$ is expanded differently between the unsorted and sorted cases: we consider only the terms of the same sort as l in the sorted translation, thus reducing the number of generated rules compared to the unsorted case.

Example 5.3. Let $S_{pz}=\mathtt{+}(\mathtt{Z},x)\twoheadrightarrow x$, $\mathcal{S}=\{\mathtt{Nat},\mathtt{Bool}\}$, and $\mathcal{F}=\mathcal{F}_{\mathtt{Nat}}\cup\mathcal{F}_{\mathtt{Bool}}$ with $\mathcal{F}_{\mathtt{Bool}}=\{\mathtt{odd}:\mathtt{Nat}\mapsto\mathtt{Bool},\ \mathtt{even}:\mathtt{Nat}\mapsto\mathtt{Bool},\ \mathtt{true}\ \mapsto\mathtt{Bool},\ \mathtt{false}\ \mapsto\mathtt{Bool}\}$. Then

$$\begin{array}{lll}\mathbb{T}_{\mathcal{S}}(S_{pz})=\{ & \varphi_{pz}(\mathtt{+}(\mathtt{Z},x))\twoheadrightarrow x, & \\ & \varphi_{pz}(\mathtt{Z})\twoheadrightarrow\bot(\mathtt{Z}), & \varphi_{pz}(\mathtt{true})\twoheadrightarrow\bot(\mathtt{true}),\\ & \varphi_{pz}(\mathtt{S}(y_1))\twoheadrightarrow\bot(\mathtt{S}(y_1)), & \varphi_{pz}(\mathtt{false})\twoheadrightarrow\bot(\mathtt{false}),\\ & \varphi_{pz}(\mathtt{+}(\mathtt{S}(y_1),y_2))\twoheadrightarrow\bot(\mathtt{+}(\mathtt{S}(y_1),y_2)), & \varphi_{pz}(\mathtt{odd}(y_1))\twoheadrightarrow\bot(\mathtt{odd}(y_1)),\\ & \varphi_{pz}(\mathtt{+}(\mathtt{+}(y_1,y_2),y_3))\twoheadrightarrow\bot(\mathtt{+}(\mathtt{+}(y_1,y_2),y_3)), & \varphi_{pz}(\mathtt{even}(y_1))\twoheadrightarrow\bot(\mathtt{even}(y_1)),\\ & \varphi_{pz}(\bot(x))\twoheadrightarrow\bot(x), & \varphi_{pz}(\bot(z))\twoheadrightarrow\bot(z)\ \}\end{array}$$

so that $x, y_1, y_2, y_3 : \texttt{Nat}, z : \texttt{Bool}$, and the signature is extended with $\{\ \bot : \texttt{Nat} \mapsto \texttt{Nat}$, $\varphi_{pz} : \texttt{Nat} \mapsto \texttt{Nat},\ \bot : \texttt{Bool} \mapsto \texttt{Bool},\ \varphi_{pz} : \texttt{Bool} \mapsto \texttt{Bool}\ \}$.

The unsorted translation $\mathbb{T}(S_{pz})$ contains the rules in $\mathbb{T}_{\mathcal{S}}(S_{pz})$ as well as the rules

$$\begin{array}{ll} \{\ \ \varphi_{pz}(\texttt{+}(\texttt{odd}(y_1), y_2)) \twoheadrightarrow \bot(\texttt{+}(\texttt{odd}(y_1), y_2)), & \varphi_{pz}(\texttt{+}(\texttt{true}, y_2)) \twoheadrightarrow \bot(\texttt{+}(\texttt{true}, y_2)), \\ \quad \varphi_{pz}(\texttt{+}(\texttt{even}(y_1), y_2)) \twoheadrightarrow \bot(\texttt{+}(\texttt{even}(y_1), y_2)), & \varphi_{pz}(\texttt{+}(\texttt{false}, y_2)) \twoheadrightarrow \bot(\texttt{+}(\texttt{false}, y_2))\ \} \end{array}$$

which operate on ill-sorted terms *w.r.t.* the sorted signature we consider here.

For the encodings of *All* and *One*, we can also exploit the profiles of the symbols and generate less rules to filter the ground terms of the original many-sorted signature. Given $f : s_1 \times \ldots \times s_n \mapsto s$, encoding produces now schemas of the form $\varphi_f(x_1 \mathbin{@} !^{s_1}\bot(\_), \ldots, x_n \mathbin{@} !^{s_n}\bot(\_), y^s) \twoheadrightarrow f(x_1, \ldots, x_n)$ and $\varphi_{f_i}(\bot(x_1^{s_1}), \ldots, \bot(x_{i-1}^{s_{i-1}}), x_i \mathbin{@} !^{s_i}\bot(\_), x_{i+1}^{s_{i+1}}, \ldots, x_n^{s_n}) \twoheadrightarrow f(x_1^{s_1}, \ldots, x_n^{s_n})$ for respectively *All* and *One* and thus generates significantly less rules than in the unsorted case where we used the exhaustive $!\bot(\_)$.

The encodings of the other operators (sequence, choice, ...) also filter terms using $x \mathbin{@} !^s\bot(\_)$, but for all $s \in \mathcal{S}$ and not for a given s, so the number of generated rules for these constructs is the same as in the unsorted case.

Example 5.4. If we consider the many-sorted signature in Example 5.1, we obtain
$\mathbb{T}_{\mathcal{S}}(All(\texttt{+}(\texttt{Z}, x) \twoheadrightarrow x)) = \mathbb{T}_{\mathcal{S}}(S_{pz}) \bigcup \{$

$$\begin{array}{ll}
\varphi_{All}(\bot(x)) \twoheadrightarrow \bot(x), & \\
\varphi_{All}(\texttt{+}(x_1, x_2)) \twoheadrightarrow \varphi_{\texttt{+}}(\varphi_{pz}(x_1), \varphi_{pz}(x_2), \texttt{+}(x_1, x_2)), & \varphi_{\texttt{+}}(Z, \texttt{+}(x_1, x_2), x) \twoheadrightarrow \texttt{+}(Z, \texttt{+}(x_1, x_2)), \\
\varphi_{\texttt{+}}(\bot(x_1), x_2, x) \twoheadrightarrow \bot(x), & \varphi_{\texttt{+}}(Z, \texttt{S}(x_1), x) \twoheadrightarrow \texttt{+}(Z, \texttt{S}(x_1)), \\
\varphi_{\texttt{+}}(x_1, \bot(x_2), x) \twoheadrightarrow \bot(x), & \varphi_{\texttt{+}}(Z, Z, x) \twoheadrightarrow \texttt{+}(Z, Z), \\
\varphi_{\texttt{+}}(\texttt{+}(x_1, x_2), \texttt{+}(y_1, y_2), x) \twoheadrightarrow \texttt{+}(\texttt{+}(x_1, x_2), \texttt{+}(y_1, y_2)), & \varphi_{All}(\texttt{S}(x_1)) \twoheadrightarrow \varphi_{\texttt{S}}(\varphi_{pz}(x_1), \texttt{S}(x_1)), \\
\varphi_{\texttt{+}}(\texttt{+}(x_1, x_2), \texttt{S}(y_1), x) \twoheadrightarrow \texttt{+}(\texttt{+}(x_1, x_2), \texttt{S}(y_1)), & \varphi_{\texttt{S}}(\bot(x_1), x) \twoheadrightarrow \bot(x), \\
\varphi_{\texttt{+}}(\texttt{+}(x_1, x_2), Z, x) \twoheadrightarrow \texttt{+}(\texttt{+}(x_1, x_2), Z), & \varphi_{\texttt{S}}(\texttt{+}(x_1, x_2), x) \twoheadrightarrow \texttt{S}(\texttt{+}(x_1, x_2)), \\
\varphi_{\texttt{+}}(\texttt{S}(x_1), \texttt{+}(y_1, y_2), x) \twoheadrightarrow \texttt{+}(\texttt{S}(x_1), \texttt{+}(y_1, y_2)), & \varphi_{\texttt{S}}(\texttt{S}(x_1), x) \twoheadrightarrow \texttt{S}(\texttt{S}(x_1)), \\
\varphi_{\texttt{+}}(\texttt{S}(x_1), \texttt{S}(y_1), x) \twoheadrightarrow \texttt{+}(\texttt{S}(x_1), \texttt{S}(y_1)), & \varphi_{\texttt{S}}(Z, x) \twoheadrightarrow \texttt{S}(Z), \\
\varphi_{\texttt{+}}(\texttt{S}(x_1), Z, x) \twoheadrightarrow \texttt{+}(\texttt{S}(x_1), Z), & \varphi_{All}(Z) \twoheadrightarrow Z, \\
\varphi_{All}(\bot(z)) \twoheadrightarrow \bot(z), & \\
\varphi_{All}(\texttt{even}(x_1)) \twoheadrightarrow \varphi_{\texttt{even}}(\varphi_{pz}(x_1), \texttt{even}(x_1)), & \varphi_{\texttt{odd}}(\bot(x_1), z) \twoheadrightarrow \bot(z), \\
\varphi_{\texttt{even}}(\bot(x_1), z) \twoheadrightarrow \bot(z), & \varphi_{\texttt{odd}}(\texttt{+}(x_1, x_2), z) \twoheadrightarrow \texttt{odd}(\texttt{+}(x_1, x_2)), \\
\varphi_{\texttt{even}}(\texttt{+}(x_1, x_2), z) \twoheadrightarrow \texttt{even}(\texttt{+}(x_1, x_2)), & \varphi_{\texttt{odd}}(\texttt{S}(x_1), z) \twoheadrightarrow \texttt{odd}(\texttt{S}(x_1)), \\
\varphi_{\texttt{even}}(\texttt{S}(x_1), z) \twoheadrightarrow \texttt{even}(\texttt{S}(x_1)), & \varphi_{\texttt{odd}}(Z, z) \twoheadrightarrow \texttt{odd}(Z), \\
\varphi_{\texttt{even}}(Z, z) \twoheadrightarrow \texttt{even}(Z), & \varphi_{All}(\texttt{false}) \twoheadrightarrow \texttt{false}, \\
\varphi_{All}(\texttt{odd}(x_1)) \twoheadrightarrow \varphi_{\texttt{odd}}(\varphi_{pz}(x_1), \texttt{odd}(x_1)), & \varphi_{All}(\texttt{true}) \twoheadrightarrow \texttt{true}\ \}
\end{array}$$

so that $x, x_1, x_2, y_1, y_2 : \texttt{Nat}, z : \texttt{Bool}$, and we extend the original signature with

$$\begin{array}{l} \{\ \varphi_{All} : \texttt{Nat} \mapsto \texttt{Nat},\ \varphi_{All} : \texttt{Bool} \mapsto \texttt{Bool},\ \bot : \texttt{Nat} \mapsto \texttt{Nat},\ \bot : \texttt{Bool} \mapsto \texttt{Bool}, \\ \quad \varphi_{pz} : \texttt{Nat} \mapsto \texttt{Nat},\ \varphi_{pz} : \texttt{Bool} \mapsto \texttt{Bool},\ \varphi_{\texttt{+}} : \texttt{Nat} \times \texttt{Nat} \times \texttt{Nat} \mapsto \texttt{Nat},\ \varphi_{\texttt{S}} : \texttt{Nat} \times \texttt{Nat} \mapsto \texttt{Nat}, \\ \quad \varphi_{\texttt{even}} : \texttt{Nat} \times \texttt{Bool} \mapsto \texttt{Bool},\ \varphi_{\texttt{odd}} : \texttt{Nat} \times \texttt{Bool} \mapsto \texttt{Bool}\ \}. \end{array}$$

The unsorted translation $\mathbb{T}(All(\texttt{+}(\texttt{Z}, x) \twoheadrightarrow x))$ would contain not only the above rules but also a significant number of rules which would be ill-sorted *w.r.t.* the many-sorted signature considered here.

We show in Section 5.3 that although the sorted translation generates less rules (modulo syntactic equivalence) than the unsorted translation, we still obtain a faithful encoding for sorted terms.

Remark 5.5. We suppose in this section that the targeted language supports overloaded symbols, but if it is not the case, like, *e.g.* in TOM, the sorted translation can be easily adapted to fit this constraint: instead of overloading the generated symbols, we add one symbol for each sort. For example, the original signature would be extended in this case with the symbol(s) $\bigcup_{s\in\mathcal{S}}\{\bot^s : s \mapsto s\}$. We then use the symbol of the appropriate sort in the rules of Figure 7, depending of the sort of its argument(s). For example, the translation of a rewrite rule $l \twoheadrightarrow r$ with $l : s_l$ becomes:

$$\mathbb{T}_{\mathcal{S}}(l \twoheadrightarrow r) = \{\ \varphi^{s_l}_{l\twoheadrightarrow r}(l) \twoheadrightarrow r\ \} \bigcup_{s\in\mathcal{S}} \{\ \varphi^{s}_{l\twoheadrightarrow r}(x \,@\, \overset{s}{!} l) \twoheadrightarrow \bot^s(x), \quad \varphi^{s}_{l\twoheadrightarrow r}(\bot^s(x^s)) \twoheadrightarrow \bot^s(x^s)\ \}$$

5.3. **Properties of the many-sorted encoding.** It is easy to see that for each of the generated rewrite rules, for any sort assignment for the variables such that the left-hand side is well-sorted, the right-hand side is also well-sorted and has the same sort as the left-hand side. Starting from this observation we can show that the reduction of a sorted term *w.r.t.* a strategy encoding is sort preserving:

Lemma 5.6 (Subject reduction)**.** *Consider a many-sorted signature $(\mathcal{S}, \mathcal{F})$, a strategy S, a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, and the term rewriting system $\mathcal{R}{=}\mathbb{T}_{\mathcal{S}}(S){\cup}\mathbb{B}_{\mathcal{S}}(\Gamma)$ built over the extended signature $(\mathcal{S}, \overline{\mathcal{F}})$. Given a term $t \in \mathcal{T}_s(\overline{\mathcal{F}})$ for some sort $s \in \mathcal{S}$, if $t \longrightarrow_{\mathcal{R}} t'$ then $t' \in \mathcal{T}_s(\overline{\mathcal{F}})$.*

Sketch. By induction on the structure of the strategy S (and the context Γ) and by case analysis on the rewrite rule applied in the reduction. For each case we consider reductions at the top position since the replacement of a sub-term by another one of the same sort is obviously sort preserving. □

As we have seen in the previous section, the rewrite rules generated by the sorted translation are the ones generated by the unsorted encoding that are well-sorted *w.r.t.* the considered many-sorted signature. Since an ill-sorted rewrite rule cannot be applied on a sorted term, the reduction of a such a sorted term is the same if we use the sorted or unsorted encoding.

Lemma 5.7 (Equivalent reductions)**.** *Consider a many-sorted signature $(\mathcal{S}, \mathcal{F})$, a strategy S, a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, and the term rewriting systems $\mathcal{R}{=}\mathbb{T}(S){\cup}\mathbb{B}(\Gamma)$ and $\mathcal{R}_{\mathcal{S}}{=}\mathbb{T}_{\mathcal{S}}(S) \cup \mathbb{B}_{\mathcal{S}}(\Gamma)$ built over the extended signature $(\mathcal{S}, \overline{\mathcal{F}})$. Given a term $t \in \mathcal{T}_{\mathcal{S}}(\overline{\mathcal{F}})$, $t \longrightarrow_{\mathcal{R}_{\mathcal{S}}} t'$ iff $t \longrightarrow_{\mathcal{R}} t'$.*

Sketch. Every rewrite rule in $\mathcal{R}_{\mathcal{S}}$ is also included in $\mathcal{R}$ and thus, if $t \longrightarrow_{\mathcal{R}_{\mathcal{S}}} t'$ then $t \longrightarrow_{\mathcal{R}} t'$. For the other direction we proceed by induction on the structure of the strategy S (and the context Γ) and by case analysis on the rewrite rule applied in the reduction.

The interesting cases concern the rule schemas using anti-patterns in the encodings of a rewrite rule, *All*, and *One*. For the first one, we remark that if a rewrite rule corresponding to the rule schema $\varphi_{l\twoheadrightarrow r}(x \,@\, \overset{s}{!} l) \twoheadrightarrow \bot(x)$ from the unsorted translation is applied to a sorted term then, there exists an identical rewrite rule corresponding to the similar rule schema from the sorted translation which can be applied. For the other two cases, we proceed similarly and use the fact that if a rewrite rule corresponding to one of the rule schemas relying on a $!\bot(\_)$ is applied, then the same rewrite rule is also exhibited by the similar rule schema using the corresponding pattern $!^s\bot(\_)$ from the sorted translation. □

The one-step reduction of a sorted term is thus exactly the same when using the sorted or unsorted encodings and since the latter is also sort preserving we can conclude that the sorted translation produces faithful strategy encodings.

Theorem 5.8 (Simulation)**.** *Given a a many-sorted signature* $(\mathcal{S}, \mathcal{F})$*, a strategy* S*, two terms* $t, t' \in \mathcal{T_S}(\mathcal{F})$*, and a context* Γ *such that* $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$*,*
(1) $\Gamma \vdash S \circ t \Longrightarrow t'$ *iff* $\mathbb{T}_{\mathcal{S}}(S) \cup \mathbb{B}_{\mathcal{S}}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow t'$*,*
(2) $\Gamma \vdash S \circ t \Longrightarrow \texttt{Fail}$ *iff* $\mathbb{T}_{\mathcal{S}}(S) \cup \mathbb{B}_{\mathcal{S}}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$

Proof. Follows immediately from Lemma 5.6, Lemma 5.7 and Theorem 3.5. □

The sorted translation can be consequently used as a strategy compiler for many-sorted languages and, as explained above, it can be used for languages allowing symbol overloading or not. If we collapse all syntactically equivalent rules in a sorted encoding we obtain less rules than in the corresponding unsorted encoding but we can still feed it into (usually unsorted) termination tools to verify the termination of the corresponding strategy for sorted terms.

6. Implementation and experimental results

The strategy translations presented in the previous sections have been implemented in a tool called StrategyAnalyser[4], written in Tom, a language that extends Java with high level constructs for pattern matching, rewrite rules and strategies (*i.e.* the tool itself is written using rules and strategies!). Given a set of rewrite rules guided by a strategy, the tool generates a plain TRS in AProVE/TTT2 syntax[5] or Tom syntax (restricted to rewrite rules only).

The tool can be configured to generate TRS at meta-level or not, in a many-sorted context or not, to use the alias notation or not, and to use the notion of anti-term or not. An encoding using anti-terms and aliasing can be directly used in a Tom program but for languages and tools which do not offer such primitives, aliases and anti-terms have to be expanded into plain rewrite rules. We explain first how this expansion is realized and we illustrate then our approach on several representative examples.

6.1. Expansion of anti-terms.

The rules given in Figure 2 can generate two kinds of rules which contain anti-terms. The first family is of the form $\varphi(\ldots, y_i \,@\, !\bot(\_), \ldots) \twoheadrightarrow u$ with $y_i \in \mathcal{X}$, and with potentially several occurrences of $!\bot(\_)$. These rules can be easily expanded into a family of rules $\varphi(\ldots, y_i \,@\, f(x_1, \ldots, x_n), \ldots) \twoheadrightarrow u$ with such a rule for all $f \in \mathcal{F}$, and with $x_1, \ldots, x_n \in \mathcal{X}$ and $n = ar(f)$. This expansion is performed recursively to eliminate all the instances of $!\bot(\_)$. The other rules containing anti-terms come from the translation of rewrite rules (**E3**) and have in the unsorted case the form $\varphi(y \,@\, !f(t_1, \ldots, t_n)) \twoheadrightarrow \bot(y)$, with $f \in \mathcal{F}^n$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If the term $f(t_1, \ldots, t_n)$ is linear, then the tool generates two families of rules:

[4] `http://github.com/rewriting/tom/tree/master/applications/strategyAnalyzer`
[5] `http://aprove.informatik.rwth-aachen.de/`

- $\varphi(g(x_1,\ldots,x_m)) \twoheadrightarrow \perp(g(x_1,\ldots,x_m))$ for all $g \in \mathcal{F}$, $g \neq f$, $x_1,\ldots,x_m \in \mathcal{X}$, $m = ar(g)$,
- $\varphi(f(x_1,\ldots,x_{i-1},x_i \mathbin{@} !t_i,x_{i+1},\ldots,x_n)) \twoheadrightarrow \perp(f(x_1,\ldots,x_n))$ for all $i \in [1,n]$ and $t_i \notin \mathcal{X}$,

with the second family of rules recursively expanded, using the same algorithm, until there is no anti-term left.

This expansion mechanism is more difficult when we want to find a convenient (finite) encoding for non-linear anti-terms, and in this case the expansion should be done, in fact, *w.r.t.* the entire translation of a rewrite rule. Given the rules $\varphi(l) \twoheadrightarrow r$ and $\varphi(y \mathbin{@} !l) \twoheadrightarrow \perp(y)$ with $l \in \mathcal{T}(\mathcal{F},\mathcal{X})$ a non-linear term, we consider the linearized version of l, denoted l', with all the variables $x_i \in \mathcal{V}ar\,(l)$ appearing more than once (m_i times, with $m_i > 1$) renamed into $z_i^1,\ldots,z_i^{m_i-1}$ (the first occurrence of x_i is not renamed). Then, these two rules can be translated into:

- $\varphi(y \mathbin{@} !l') \twoheadrightarrow \perp(y)$
- $\varphi(l') \twoheadrightarrow \varphi'(l', x_1 = z_1^1 \wedge \cdots \wedge x_1 = z_1^{m_1-1} \wedge \cdots \wedge x_n = z_n^1 \wedge \cdots \wedge x_n = z_n^{m_n-1})$
- $\varphi'(l', \mathrm{tt}) \twoheadrightarrow r$
- $\varphi'(l', \mathrm{ff}) \twoheadrightarrow \perp(l')$

with the first rule containing now the linear anti-term $!l'$ expanded as previously. The rules generated for equality and conjunction are as expected.

The expansion goes in the same way in the many-sorted case, but using only symbols of the appropriate sort; a sort $\mathbb{B}$, an overloaded equality symbol $\bigcup_{s\in\mathcal{S}}\{= \,: s \times s \mapsto \mathbb{B}\}$ together with the symbols $\{\mathrm{tt}\colon\ \mapsto \mathbb{B},\ \mathrm{ff}\colon\ \mapsto \mathbb{B},\ \wedge\colon \mathbb{B}\times\mathbb{B} \mapsto \mathbb{B}\}$ are added to the extended signature. The rules generated for equality are then restricted to those whose left-hand side and right-hand side have the same sort. Given a many-sorted signature $(\mathcal{S},\mathcal{F})$, this set of rules consists of $\bigcup_{s\in\mathcal{S}}\bigcup_{f\in\mathcal{F}_s^n}\{\ f(x_1^{s_1},\ldots,x_n^{s_n}) = f(y_1^{s_1},\ldots,y_n^{s_n}) \twoheadrightarrow x_1^{s_1} = y_1^{s_1} \wedge\ldots\wedge x_n^{s_n} = y_n^{s_n} \wedge \mathrm{tt}\ \}$ $\bigcup_{s\in\mathcal{S}}\bigcup_{f\in\mathcal{F}_s^n}\bigcup_{g\neq f\in\mathcal{F}_s^n}\{\ f(x_1^{s_1},\ldots,x_n^{s_n}) = g(y_1^{s_1},\ldots,y_m^{s_m}) \twoheadrightarrow \mathrm{ff}\ \}$. For the translation $\mathbb{T}_{\mathcal{M}}$ in Section 4, the expansion is performed as in the unsorted case, except that the expansion process generates meta-level representations directly.

6.2. **Examples.** The first example we consider in what follows comes from an optimizer for multi-core architectures [11], a project where abstract syntax trees are manipulated and transformations are expressed using rewrite rules and strategies, and consists of two rewrite rules identified as patterns occurring often in various forms in the project. First, the rewrite rule $g(f(x)) \twoheadrightarrow f(g(x))$ corresponds to the search for an operator g (which can have more than one parameter in the general case) which is pushed down under another operator f (again, this operator may have more than one parameter). This rule is important since the corresponding (innermost) reduction of a term of the form $t_{\mathrm{gf}} = \overbrace{g(\underbrace{f(\cdots(f}_{n}(g(\underbrace{f(\cdots(f}_{n}(g(\underbrace{f(\cdots(f}_{n}(g}^{m}(a)))\cdots)$, with, for example, $n = 10$ and $m = 18$ occurrences of g, involves $O(n^2m^2)$ computations and could be a performance bottleneck.

Second, the rewrite rule $h(x) \twoheadrightarrow g(h(x))$ corresponds to wrapping some parts of a program by some special constructs, like `try/catch` for example, and it is interesting since its uncontrolled application is obviously non-terminating.

At present, strategy definitions given as input to StrategyAnalyser are written in a simple functional style.

Example 6.1. The syntax allowing the definition of the above rewrite rules and possible corresponding strategies could be defined as follows:

```
abstract syntax
  T = a() | b() | f(T) | g(T) | h(T)
strategies
  gfx()       = [ g(f(x)) -> f(g(x)) ]
  hx()        = [ h(x) -> g(h(x)) ]
  obu(t)      = mu x.(one(x) <+ t)              # obu stands for OnceBottomUp
  bu(t)       = mu x.(all(x) ; (t <+ Identity)) # bu stands for BottomUp
  repeat(s)   = mu y.((s ; y) <+ Identity)      # naive definition of innermost
  mainStrat() = repeat(obu(gfx()))              # strategy to compile
```

As a second example, we consider a strategy involving rewrite rules which are either non left-linear or non right-linear and which are non-terminating if their application is not guided by a strategy.

Example 6.2. We consider the following rewrite rules which implement the distributivity and factorization of symbolic expressions composed of `+` and `*` and their application under a specific strategy:

```
abstract syntax
  T = Plus(T,T) | Mult(T,T) | Val(V)
  V = a() | b()
strategies
  dist()       = [ Mult(x,Plus(y,z)) -> Plus(Mult(x,y),Mult(x,z)) ]
  fact()       = [ Plus(Mult(x,y),Mult(x,z)) -> Mult(x,Plus(y,z)) ]
  innermost(s) = mu x.(all(x) ; ((s ; x) <+ Identity))
  mainStrat()  = innermost(dist()) ; innermost(fact())
```

As a third example, we consider a larger program inspired by the Tom compiler itself, where the signature is composed of seven sorts and contains a significant number of constructors.

Example 6.3. We consider two rewrite rules. The purpose of the first one, `compile`, is to identify `Match` constructs and to replace them by instructions of the form `If`, `Assign`, `WhileDo`, ..., which implement the matching algorithm. In the Tom compiler, these instructions are transformed by the backend into executable code written in Java or `C` for instance. The second rule identifies occurrences of variables and produces a term which contains both the new variable and the initial one (this rule represents the first stage of a refactoring process).

```
abstract syntax # refactor example
  CodeList = NilCode() | ConsCode(Code,CodeList)
  Code = Match(TermList) | Assign(Name,Exp) | If(Exp,Code,Code) | WhileDo(Exp,Code)
       | Nop() | ...
  Exp  = Or(Exp,Exp) | And(Exp,Exp) | IsFsym(Name,Term) | EqualTerm(Term,Term)
       | TrueTL() | FalseTL() | ...
  TermList = ConsTerm(Term,TermList) | NilTerm()
  Term = VarTerm(Name) | ApplTerm(Name,TermList) | RenamedTerm(Term,Term)
  Nat = Z() | S(Nat)
  Name = Name(Nat)
strategies
  compile() = [ Match(l) -> <...Code...> ]
```

```
  rename() = [ VarTerm(Name(n)) -> RenamedTerm(VarTerm(Name(S(n))), VarTerm(Name(n))) ]
  td(s) = mu x.((s <+ Identity) ; all(x))      # td stands for TopDown
  tdstoponsucces(s) = mu x.(s <+ all(x))
  mainStrat() = tdt(compile()) ; tdstoponsucces(rename())
```

The right-hand side of the rule `rename` contains the left-hand side of the rule and thus a top-down strategy on this rule would not be terminating. Furthermore, it would rename the second argument of `RenamedTerm` that we want to keep unchanged. The strategy we consider (`tdstoponsucces`) is interesting because it searches for a `VarName` constructor in a top-down way, but performs the replacement only once and does not continue the search into sub-terms when a transformation is performed. Intuitively, this strategy is terminating, and we will see that termination tools are able to prove it.

We also consider a relatively large example containing 4 sorts, 13 constructors, and 20 rules.

Example 6.4. The following represents the implementation of red-black-trees based on [33], but expressed using rules and strategies.

```
abstract syntax # rbTree example
  Tree = E() | T(Color,Tree,Nat,Tree) | balance(Tree) | ins(Nat,Tree)
       | insAux(Nat,Tree,Cmp)
  Color = R() | B()
  Nat = Z() | S(Nat)
  Cmp = lt() | gt() | eq0() | cmp(Nat,Nat)
strategies
  b1() = [ balance(T(B(),T(R(),T(R(),a1,a2,a3),x,b),y,T(R(),c,z,d))) ->
           T(R(),T(B(),T(R(),a1,a2,a3),x,b),y,T(B(),c,z,d)) ]
  b2() = [ balance(T(B(),T(R(),a,x,T(R(),b1,b2,b3)),y,T(R(),c,z,d))) ->
           T(R(),T(B(),a,x,T(R(),b1,b2,b3)),y,T(B(),c,z,d)) ]
  ... # rules b3()...b8()
  b9() = [ balance(t) -> t ] # no balancing necessary
  ... # rules i1()...i5() and c1()...c4()
  mainStrat() = repeat(obu(b1() <+ b2() <+ b3() <+ ...))
```

The complexity here comes also from the presence of a constructor `T` of arity 4, whose negation ($!T(\ldots)$) generates a large list of patterns to capture the cases where the rule cannot be applied. The anti-term $!balance(T(B(), T(R(), T(R(), a1, a2, a3), x, b), y, T(R(), c, z, d)))$ is expanded into 108 patterns.

6.3. **Generation of TRS for termination analysis.** When run with the flag `-aprove`, the StrategyAnalyser tool generates a TRS in AProVE/TTT2 syntax which can be analyzed by any tool accepting this syntax. In this case, aliases and anti-terms are always completely expanded leading generally to a significant number of plain rewrite rules.

The tool can be configured to generate many-sorted TRS or to generate the meta-level representation of the TRS. The number of generated rules for a strategy could thus vary a lot. In Table 1, we give for each example the number of generated rules in the (U)nsorted case, in the many-(S)orted case, and in the (M)eta-level case. The last column indicates whether the termination of the generated TRS has been proven or disproven by AProVE.

In practice, AProVE is able to handle relatively big sets of rules and, for example, the termination of the strategy repeat(obu(gfx)), which is translated into 91 rules, is proven

Name	Strategy	U	S	M	APROVE
repeat(dist)	$\mu X \,.\, ((\mathsf{dist} \,;\, X) \nleftarrow \mathit{Identity})$	49	57	25	✓
repeat(fact)	$\mu X \,.\, ((\mathsf{fact} \,;\, X) \nleftarrow \mathit{Identity})$	84	78	60	✓
repeat(dist ; fact)	$\mu X \,.\, (((\mathsf{dist} \,;\, \mathsf{fact}) \,;\, X) \nleftarrow \mathit{Identity})$	110	107	77	✗
td(dist)	$\mu X \,.\, ((\mathsf{dist} \nleftarrow \mathit{Identity}) \,;\, \mathit{All}(X))$	97	68	35	✓
obu(fact)	$\mu X \,.\, (\mathit{One}(X) \nleftarrow \mathsf{fact})$	102	83	70	✓
repeat(obu(fact))	$\mu X \,.\, ((\mathsf{obu}(\mathsf{fact}) \,;\, X) \nleftarrow \mathit{Identity})$	138	125	82	✓
factorize	$\mu X \,.\, (\mathit{All}(X) \,;\, ((\mathsf{fact} \,;\, \mathit{All}(X)) \nleftarrow \mathit{Identity}))$	162	124	80	✓
simplify	td(dist) ; factorize	272	206	110	✓
innermost(dist)	$\mu X \,.\, (\mathit{All}(X) \,;\, ((\mathsf{dist} \,;\, X) \nleftarrow \mathit{Identity}))$	127	103	45	✓
innermost(fact)	$\mu X \,.\, (\mathit{All}(X) \,;\, ((\mathsf{fact} \,;\, X) \nleftarrow \mathit{Identity}))$	162	124	80	✓
repeat(td(dist))	$\mu X \,.\, ((\mathsf{td}(\mathsf{dist}) \,;\, X) \nleftarrow \mathit{Identity})$	133	110	47	✗
bu(hx)	$\mu X \,.\, (\mathit{All}(X) \,;\, (\mathsf{hx} \nleftarrow \mathit{Identity}))$	51	51	31	✓
td(hx)	$\mu X \,.\, ((\mathsf{hx} \nleftarrow \mathit{Identity}) \,;\, \mathit{All}(X))$	51	51	31	✗
repeat(obu(gfx))	$\mu X \,.\, ((\mathsf{obu}(\mathsf{gfx}) \,;\, X) \nleftarrow \mathit{Identity})$	91	91	47	✓
innermost(gfx)	$\mu X \,.\, (\mathit{All}(X) \,;\, ((\mathsf{gfx} \,;\, X) \nleftarrow \mathit{Identity}))$	85	85	45	✓
propagate	$\mu X \,.\, (\mathsf{gfx} \,;\, (\mathit{All}(X) \nleftarrow \mathit{Identity}))$	73	73	41	✓
bu(propagate)	$\mu X \,.\, (\mathit{All}(X) \,;\, (\mathsf{propagate} \nleftarrow \mathit{Identity}))$	127	127	58	✓
bu(propagate2)	– bu(propagate)with `f`, `g`, `h` of arity 2 –	991	378	66	✓
refactor	td(compile) ; tdStopOnSuccess(rename)	63065	2350	145	∅ ✓ ✓
rbTree	$\mu X \,.\, ((\mathsf{obu}(\mathsf{b1} \nleftarrow \mathsf{b2} \nleftarrow \cdots) \,;\, X) \nleftarrow \mathit{Identity})$	1956	1449	1260	∅

Table 1: Termination analysis: the columns U, S, and M indicate the number of plain rewrite rules generated for the strategy, respectively in the (U)nsorted case, many-(S)orted case, and (M)eta-level case. The column APROVE indicates (for the U, S and M cases) whether the termination of the rules has been proven (✓) or disproven (✗) by APROVE; ∅ is used when APROVE gives no information. A unique symbol in the row indicates that the results are the same for the three cases.

in approximately 10 s (using the web interface). Similarly, for the example bu(propagate2) which corresponds to an extension of the example bu(propagate) where symbols `f`, `g`, and `h` become binary and the rule gfx is replaced by $g(f(x,y),z) \rightarrowtriangle f(g(x,z),y)$, we generate 991 rules and the proof can be done in less then 80 s. For this example, when considering the 378 many-sorted rules, the proof can be done in less than 15 s and in the meta-level case, consisting of 66 rules, the proof can be done in approximately 12 s. The size of the (left-hand and right-hand sides of the) rules seems to be an important factor since the termination for the rbTree example consisting of roughly 1000 rules cannot be (dis)proven while the 2350 rules of the many-sorted encoding of refactor can be handled in 160 s.

The termination of some strategies like, for example, repeat(obu(gfx)) might look pretty easy to show for an expert, but termination is less obvious for more complex strategies like, for example, bu(propagate), which is a specialized version of repeat(obu(gfx)), or rbu(fact), which is a variant of bu(fact).

The approach was effective not only in proving termination of some strategies, but also in disproving it when necessary. Once again this might look obvious for some strategies like, for example, td(hx), which involves a non-terminating rewrite rule, but it is less clear for strategies combining terminating rewrite rules or strategies like, *e.g.*, repeat(dist;fact).

6.4. **Generation of executable TRS.** When run with the flag `-tom`, the STRATEGY-ANALYSER tool generates a TRS in TOM syntax which can be subsequently compiled into Java code and executed.

By default, TOM executes a plain TRS with a built-in leftmost-innermost strategy encoded using function calls. But TOM can also execute a rule controlled by a user-defined strategy. In that case, the user-defined strategy is encoded into Java objects and is evaluated using a library written in Java. This library provides several implementations where the notion of failure can be encoded by a Java exception or by a special value to provide an exception-free implementation.

Name	TRS	Meta TRS	TOM	TOM*
repeat(dist)	< 5	< 5	< 5	< 5
repeat(fact)	< 5	< 5	13	< 5
obu(fact)	< 5	< 5	< 5	< 5
repeat(obu(fact))	190	323	2460	120
innermost(dist)	332	347	650	230
innermost(fact)	310	472	308	149
bu(hx)	< 5	< 5	< 5	< 5
repeat(obu(gfx))	400	780	6300	414
innermost(gfx)	553	433	4180	365
propagate	< 5	< 5	< 5	< 5
bu(propagate)	49	108	46	42
refactor	-	570	220	115
rbTree	1480	2200	2070	840

Table 2: Benchmarks: the column TRS indicates the execution time in milliseconds for the generated TRS compiled using TOM (*i.e.* using a built-in leftmost-innermost strategy), the column Meta TRS indicates the execution time for a meta-level TRS compiled using TOM, the column TOM indicates the execution time of the same strategy written directly in TOM, using Java exception-based implementation, and the column TOM* indicates the execution time of the TOM Java exception-free implementation.

It is interesting to see how the varying number of generated rules for a strategy impacts the efficiency of the execution of such a system. If we execute a TOM+Java program corresponding to the repeat(obu(gfx)) strategy with a classic built-in implementation where strategy failure is implemented by a Java exception, the normalization of the term t_{gf} takes 6.3 s[6] (Table 2, column TOM). With an alternative built-in implementation which uses a special value and does not throw Java exceptions, the computation time decreases to 0.41 s (Table 2, column TOM*). The strategy repeat(obu(gfx)) is translated into an executable TRS containing 91 TOM plain rewrite rules and the normalization takes in this case 0.4 s! When generating a meta-level TRS, the number of rules decrease to 47, but the normalization takes 0.78 s. When implementing the innermost(gfx) strategy natively in TOM Java, using a built-in implementation with a special encoding of failure, the normalization takes 0.36 s. The same strategy is translated into an executable TRS which contains 85 plain rewrite

[6]on a MacPro 3GHz

rules in the unsorted case and 45 rules in the meta-level case. The first TRS normalizes the term t_{gf} in 0.55 s, whereas it takes 0.43 s to the meta-level TRS to normalize the term. This example is interesting because it shows that the meta-level approach allows to considerably reduce the number of rules, and thus the size of the generated code, without slowing down the execution time. The performances are confirmed for the large examples; the 63065 rules generated for refactor with the plain encoding could not be compiled with Tom.

We observe that, although the number of generated rules could be significant, the execution times of the resulting plain TRS are comparable to those obtained with the native implementation of Tom strategies. This might look somewhat surprising but can be explained when we take a closer look to the way rewriting rules and strategies are generally implemented:

- the implementation of a TRS can be done in an efficient way since the complexity of syntactic pattern matching depends only on the size of the term to reduce and, thanks to many-to-one matching algorithms [21, 12], the number of rules has almost no impact.
- in Tom, each native strategy constructor is implemented by a Java class with a `visit` method which implements (*i.e.* interprets) the semantics of the corresponding operator. The evaluation of a strategy `S` on a term `t` is implemented thus by a call `S.visit(t)` and an exception (`VisitFailure`) is thrown when the application of a strategy fails.

In the generated TRS, the memory allocation involved in the construction of terms headed by the $\perp$ symbol encoding failure appears to be more efficient than the costly Java exception handling. This is reflected by better performances of the plain TRS implementation compared to the exception-based native implementation (especially when the strategy involves a lot of failures). We obtain performances with the generated TRS comparable to an exception-free native implementation of strategies (as we can see with the columns TRS and Tom* in Table 2), because efficient normalization techniques can be used for the plain TRS, since its rewrite rules are not controlled by a programmable strategy.

7. Conclusions and further work

We have proposed a translation of programmable strategies into plain rewrite rules that we have proven sound and complete; Figure 5 summarizes the obtained results. Well-established termination methods can be thus used to (dis)prove the termination of the obtained TRS and we can deduce, as a direct consequence, the property for the corresponding strategy. Alternatively, the translation can be used as a strategy compiler for languages which do not implement natively such primitives.

The translation has been also adapted to cope with many-sorted signatures, and although the size of the obtained encodings could be smaller than in the unsorted case, it still depends strongly on the underlying signature. We have proposed a meta-level representation of the terms and a corresponding translation which produces encodings whose size depends to a lesser extent on the signature and are significantly smaller than the ones obtained with the (un)sorted translation.

The translation has been implemented in Tom and can generate, for the moment, plain TRS using either a Tom or an AProVE/TTT2 syntax. We have experimented with classic strategies and AProVE and TTT2 have been able to (dis)prove the termination even when the number of generated rules was significant. The performances for the generated executable TRS are comparable to the ones of the Tom built-in (exception-free) strategies.

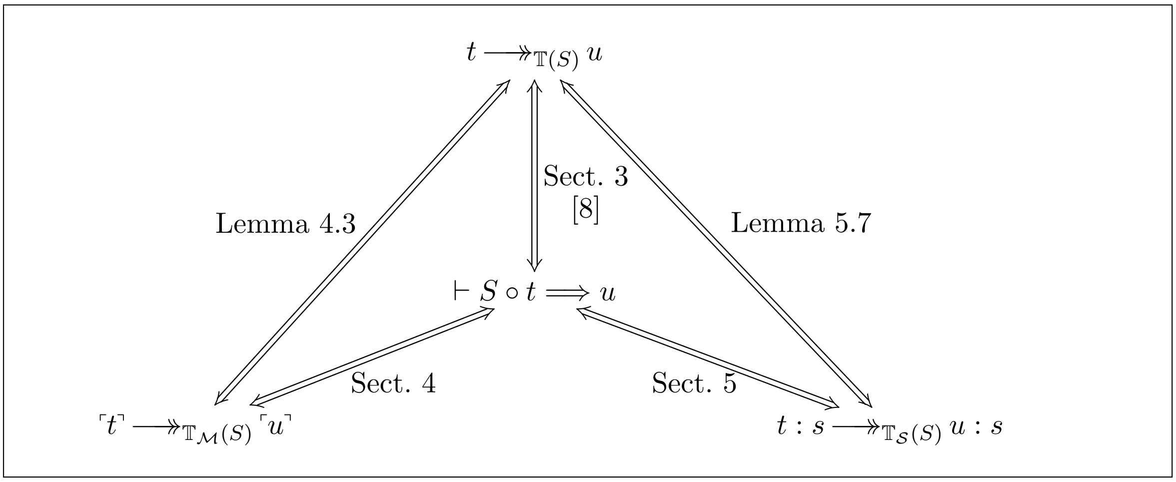

Figure 5: Relationships between the different encodings of this paper

The framework can be of course improved. When termination is disproven and a counterexample can be exhibited, it is interesting to reproduce the corresponding infinite reductions in terms of strategy derivations. Since the TRS reductions corresponding to distinct (sub-)strategy derivations are not interleaved, we think that, when the infinite reduction starts with a term headed by the symbol encoding the strategy, the back-translation of the counterexample provided by the termination tools can be automatized. When the counterexample concerns another symbol than the one encoding the strategy we could try to rebuild a complete infinite reduction by a backward application of the rules in the encoding until an appropriate term (headed by the symbol encoding the strategy) is found; although such an approach wouldn't work in all the cases it could give valuable warnings concerning the design of the strategy under investigation.

As far as the executable TRS is concerned, we intend to develop new backends allowing the integration of programmable strategies in other languages than Tom.

Acknowledgments. We would like to thank the anonymous referees for their valuable comments and suggestions that led to a substantial improvement of the paper.

References

[1] Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas. Context-sensitive dependency pairs. *Information and Computation*, 208(8):922–968, 2010.

[2] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(12):133 – 178, 2000.

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

[4] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking rewriting on java. In *RTA '07*, volume 4533 of *LNCS*, pages 36–47. Springer-Verlag, 2007.

[5] Emilie Balland, Pierre-Etienne Moreau, and Antoine Reilles. Effective strategic programming for Java developers. *Software: Practice and Experience*, 44(2):129–162, 2012.

[6] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In *WRLA '98*, volume 15. ENTCS, 1998.

[7] Horatiu Cirstea, Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-patterns for rule-based languages. *Journal of Symbolic Computation*, 45(5):523 – 550, 2010. Symbolic Computation in Software Science.

[8] Horatiu Cirstea, Sergueï Lenglet, and Pierre-Etienne Moreau. A faithful encoding of programmable strategies into term rewriting systems. In *RTA '15*, volume 36 of *LIPIcs*, pages 74–88. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
[9] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The Maude 2.0 System. In *RTA '03*, volume 2706 of *LNCS*, pages 76–87. Springer-Verlag, 2003.
[10] Jörg Endrullis and Dimitri Hendriks. From outermost to context-sensitive rewriting. In *RTA '09*, volume 5595 of *LNCS*, pages 305–319. Springer, 2009.
[11] Yuki Tomofumi et al. Alphaz: A system for design space exploration in the polyhedral model. Technical Report CS-12-101, Colorado State University, 2012.
[12] Fabrice Le Fessant and Luc Maranget. Optimizing pattern matching. In *ICFP '01*, pages 26–37. ACM Press, 2001.
[13] Olivier Fissore, Isabelle Gnaedig, and Hélène Kirchner. Simplification and termination of strategies in rule-based languages. In *PPDP '03*, pages 124–135. ACM, 2003.
[14] Carsten Fuhs, Jürgen Giesl, Michael Parting, Peter Schneider-Kamp, and Stephan Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2):133–160, 2011.
[15] Jürgen Giesl and Aart Middeldorp. Innermost termination of context-sensitive rewriting. In *DLT '02*, volume 2450 of *LNCS*, pages 231–244. Springer, 2002.
[16] Jürgen Giesl and Aart Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14(4):379–427, 2004.
[17] Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated termination proofs for haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2):7, 2011.
[18] Jürgen Giesl, René Thiemann, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37:2006, 2006.
[19] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.
[20] Isabelle Gnaedig and Hélène Kirchner. Termination of rewriting under strategies. *ACM Transactions on Computational Logic*, 10(2), 2009.
[21] Albert Gräf. Left-to-right tree pattern matching. In *RTA '91*, volume 488 of *LNCS*, pages 323–334. Springer-Verlag, April 1991.
[22] Nao Hirokawa and Aart Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(12):172 – 199, 2005.
[23] B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.
[24] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, 1991.
[25] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.
[26] Markus Kaiser and Ralf Lämmel. An Isabelle/HOL-based model of stratego-like traversal strategies. In *PPDP '09*, pages 93–104. ACM, 2009.
[27] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In *RTA '09*, volume 5595 of *LNCS*, pages 295–304. Springer-Verlag, 2009.
[28] Ralf Lämmel, Simon J. Thompson, and Markus Kaiser. Programming errors in traversal programs over structured data. *Science of Computer Programing*, 78(10):1770–1808, 2013.
[29] María Manzano. Many-sorted logic and its applications. chapter Introduction to Many-sorted Logic, pages 3–86. John Wiley & Sons, Inc., New York, NY, USA, 1993.
[30] José Meseguer and Christiano Braga. Modular rewriting semantics of programming languages. In *Algebraic Methodology and Software Technology*, volume 3116 of *LNCS*, pages 364–378. Springer Berlin Heidelberg, 2004.
[31] Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A Pattern Matching Compiler for Multiple Target Languages. In *CC '03*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, 2003.
[32] Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.

[33] Chris Okasaki. Red-black trees in a functional setting. *Journal of Functional Programming*, 9(4):471–477, 1999.

[34] Matthias Raffelsieper and Hans Zantema. A transformational approach to prove outermost termination automatically. *ENTCS*, 237:3–21, 2009.

[35] Grigore Rosu and Traian-Florin Serbanuta. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.

[36] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds.

[37] René Thiemann. From outermost termination to innermost termination. In *SOFSEM '09*, volume 5404 of *LNCS*, pages 533–545. Springer, 2009.

[38] René Thiemann and Aart Middeldorp. Innermost termination of rewrite systems by labeling. *ENTCS*, 204:3–19, 2008.

[39] Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In *RTA '01*, volume 2051 of *LNCS*, pages 357–361. Springer-Verlag, 2001.

[40] Eelco Visser. A survey of strategies in rule-based program transformation systems. *Journal of Symbolic Computation*, 40(1):831 – 873, 2005.

[41] Eelco Visser, Zine-el-Abidine Benaissa, and Andrew Tolmach. Building program optimizers with rewriting strategies. In *ICFP '98*, pages 13–26. ACM Press, 1998.

[42] Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Incorporated, 5 edition, 2003.

8. Proofs for the Regular Translation

Lemma 8.1 (Propagation lemma). *Let $t \in \mathcal{T}(\mathcal{F})$, S, and Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$. We have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S) \bullet \varphi_S(\bot(t)) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$, and if $\mathbb{B}(\Gamma) \cup \mathbb{T}(S) \bullet \varphi_S(\bot(t)) \longrightarrow\!\!\!\!\!\rightarrow u$, then $u = \bot(t)$.*

Proof. For all $S \neq X$, the only rule in $\mathbb{B}(\Gamma) \cup \mathbb{T}(S)$ that can rewrite $\varphi_S(\bot(t))$ is $\varphi_S(\bot(t)) \twoheadrightarrow \bot(t)$. For $S = X$, because $X \in \mathcal{D}om_{\mathcal{X}}(\Gamma)$, $\mathbb{B}(\Gamma)$ contains the rule $\varphi_X(\bot(t)) \twoheadrightarrow \bot(t)$, and it is also the only rule that can rewrite $\varphi_X(\bot(t))$. □

Lemma 8.2 (Rigid). *Given a term $t \in \mathcal{T}(\mathcal{F})$, a strategy S, and a context Γ, the term t is in normal form w.r.t. $\mathbb{B}(\Gamma) \cup \mathbb{T}(S)$. If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow u$, then $S = X$ and $X \in \mathcal{D}om_{\mathcal{X}}(\Gamma)$, or $\mathbb{T}(S) \subseteq \mathbb{B}(\Gamma) \cup \mathbb{T}(S')$.*

Proof. A term $t \in \mathcal{T}(\mathcal{F})$ does not contain any φ symbols, and all the rules in $\mathbb{B}(\Gamma) \cup \mathbb{T}(S)$ assumes a φ symbol at the root, hence the first result holds.

Because of this result, if $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow u$, then $\mathbb{B}(\Gamma) \cup \mathbb{T}(S')$ contains a rule of the form $\varphi_S(\_) \twoheadrightarrow \ldots$. If $S \neq X$, then it is possible only if Γ or S' contains S, and then it is easy to prove that $\mathbb{T}(S) \subseteq \mathbb{B}(\Gamma) \cup \mathbb{T}(S')$.

If $S = X$, then only the translation of contexts or recursive strategies generate rules than can rewrite a term of the form $\varphi_X(t)$. But according to the Barendregt convention, any recursive strategy in Γ or S' will be of the form $\mu Y.S''$, with $Y \neq X$. The only remaining possibility is $\mathbb{B}(\Gamma) \bullet \varphi_X(t) \longrightarrow\!\!\!\!\!\rightarrow u$, which is possible only if Γ binds X. □

Lemma 8.3. *If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow u$, then $t \in \mathcal{T}(\mathcal{F})$ or $t = \bot(\_)$.*

Proof. Immediate by definition of the translation. □

Lemma 8.4 (Initial term as failure). *Given a term $t \in \mathcal{T}(\mathcal{F})$, if $\mathbb{B}(\Gamma) \cup \mathbb{T}(S) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t')$ then $t = t'$.*

Proof. By induction on the length of the reduction $\mathbb{B}(\Gamma) \cup \mathbb{T}(S) \bullet \varphi(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t')$.

If $S = \mathit{Identity}$, it is not possible to obtain $\bot(t')$ from $\varphi_{\mathit{Identity}}(t)$, because $t \in \mathcal{T}(\mathcal{F})$.

If $S = \mathit{Fail}$, then because $t \in \mathcal{T}(\mathcal{F})$, the only rule that can be applied to $\varphi_{\mathit{Fail}}(t)$ is $\varphi_{\mathit{Fail}}(x \,@\, !\bot(\_)) \twoheadrightarrow \bot(x)$, and we obtain $\bot(t)$, as wished.

If $S = l \twoheadrightarrow r$, then the only rule that can be applied to $\varphi_{l \twoheadrightarrow r}(t)$ (with $t \in \mathcal{T}(\mathcal{F})$) to produce $\bot(t')$ is $\varphi_{l \twoheadrightarrow r}(x \,@\, !l) \twoheadrightarrow \bot(x)$, and we obtain $\bot(t)$, as required.

Suppose $S = S_1; S_2$. Because $\mathbb{T}(S_1)$ and $\mathbb{T}(S_2)$ cannot reduce $\varphi_{S_1;S_2}(t)$ (by Lemma 8.2), the first rule to be applied is $\varphi_{S_1;S_2}(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_;(\varphi_{S_2}(\varphi_{S_1}(x)), x)$, and we obtain $\varphi_;(\varphi_{S_2}(\varphi_{S_1}(t)), t)$. By Lemma 8.3, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2)$ cannot reduce $\varphi_{S_2}(\varphi_{S_1}(t))$, and by Lemma 8.2, only $\mathbb{T}(S_1)$ (or $\mathbb{B}(\Gamma)$ is $S_1 = X$) can reduce $\varphi_{S_1}(t)$. We have several possibilities. If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(u)$, then by induction, $u = t$, and by Lemma 8.1, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_;(\varphi_{S_2}(\bot(t)), t) \longrightarrow\!\!\!\!\!\rightarrow \varphi_;(\bot(t), t)$. The only rule that can be applied to the latter term is $\varphi_;(\bot(\_), x) \twoheadrightarrow \bot(x)$, and we obtain $\bot(t)$, as wished. Otherwise, we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow u$. Because we obtain $\bot(t')$ as a result, necessarily the rule $\varphi_;(\bot(\_), x) \twoheadrightarrow \bot(x)$ has been applied, which means that $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(u) \longrightarrow\!\!\!\!\!\rightarrow \bot(t'')$ for some t''. By induction, $t'' = u$, and consequently, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_;(\varphi_{S_2}(u), t) \longrightarrow\!\!\!\!\!\rightarrow \varphi_;(\bot(u), t)$. We then have $\{\varphi_;(\bot(\_), x) \twoheadrightarrow \bot(x)\} \bullet \varphi_;(\bot(u), t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$, as required.

Suppose $S = S_1 \leftrightarrow\!\!\!\!+ S_2$. Because $\mathbb{T}(S_1)$ and $\mathbb{T}(S_2)$ cannot reduce $\varphi_{S_1 \leftrightarrow\!\!\!\!+ S_2}(t)$ (by Lemma 8.2), the first rule to be applied is $\varphi_{S_1 \leftrightarrow\!\!\!\!+ S_2}(x \,@\, !\bot(\_)) \twoheadrightarrow \varphi_{\leftrightarrow\!\!\!\!+}(\varphi_{S_1}(x))$, and we obtain $\varphi_{\leftrightarrow\!\!\!\!+}(\varphi_{S_1}(t))$. By Lemma 8.2, only $\mathbb{T}(S_1)$ (or $\mathbb{B}(\Gamma)$ is $S_1 = X$) can reduce $\varphi_{S_1}(t)$. We have two possibilities. If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(u)$, then by induction $u = t$, and then $\{\varphi_{\leftrightarrow}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x)\} \bullet \varphi_{\leftrightarrow}(\bot(t)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{S_2}(t)$. By Lemma 8.2, only $\mathbb{T}(S_2)$ (or $\mathbb{B}(\Gamma)$ is $S_2 = X$) can reduce $\varphi_{S_2}(t)$, therefore we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t')$. By induction, we have $t = t'$, and the result holds. Otherwise, we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow u$ with $u \in \mathcal{T}(\mathcal{F})$. But then, the only rule that can be applied to $\varphi_{\leftrightarrow}(u)$ is $\varphi_{\leftrightarrow}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x$, and we obtain $u \in \mathcal{T}(\mathcal{F})$ as a result of the reduction of $\varphi_{S_1 \leftrightarrow S_2}(t)$, which is in contradiction with the original hypothesis.

Suppose $S = \mu X.S'$. Because $\mathbb{T}(S')$ cannot reduce $\varphi_{\mu X.S'}(t)$ (by Lemma 8.2), the first rule to be applied is $\varphi_{\mu X.S'}(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y)$, and we obtain $\varphi_{S'}(t)$. We therefore have $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t')$ with less steps than the original reduction. Besides, only $\mathbb{T}(S')$ (or $\mathbb{B}(\Gamma)$ if $S' = X$) can reduce $\varphi_{S'}(t)$ (Lemma 8.2), therefore we have in fact $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t')$. By induction, $t' = t$, and the result holds.

Suppose $S = X$. By Lemma 8.2, only $\mathbb{B}(\Gamma)$ can reduce $\varphi_X(t)$, and assuming $X\colon S'$ belongs to Γ, the only rule that can be applied is $\varphi_X(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y)$, which generates $\varphi_{S'}(t)$. Only $\mathbb{T}(S')$ (or $\mathbb{B}(\Gamma)$ if $S' = Z$) can reduce $\varphi_{S'}(t)$ (Lemma 8.2), therefore we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t')$ with less steps than the original reduction. By induction, $t' = t$, and the result holds.

Suppose $S = \mathit{All}(S')$. If t is a constant c, then we cannot obtain $\bot(t')$. Suppose $t = f(t_1, \ldots, t_n)$. Because $\mathbb{T}(S')$ cannot reduce $\varphi_{\mathit{All}(S')}(t)$ (by Lemma 8.2), the first rule to be applied is $\varphi_{\mathit{All}(S)}(f(x_1, \ldots, x_n)) \twoheadrightarrow \varphi_f(\varphi_{S'}(x_1), \ldots, \varphi_{S'}(x_n), f(x_1, \ldots, x_n))$ to generate $\varphi_f(\varphi_{S'}(t_1), \ldots, \varphi_{S'}(t_n), f(t_1, \ldots, t_n))$. Note that according to Lemma 8.2, the rules in $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mathit{All}(S'))$ cannot rewrite $f(t_1, \ldots, t_n)$ in $\varphi_f(\varphi_{S'}(t_1), \ldots, \varphi_{S'}(t_n), f(t_1, \ldots, t_n))$. From this term, the only possibility to obtain $\bot(t')$ is to apply one of the rules of the form $\varphi_f(\_, \ldots \bot(\_), \ldots, \_, z) \twoheadrightarrow \bot(z)$, which generates $f(t_1, \ldots, t_n)$, as wished.

Suppose $S = \mathit{One}(S')$. If t is a constant c, then the only rule that can be applied is $\varphi_{\mathit{One}(S')}(c) \twoheadrightarrow \bot(c)$, hence the result holds. Suppose $t = f(t_1, \ldots, t_n)$. Because $\mathbb{T}(S')$ cannot reduce $\varphi_{\mathit{One}(S')}(t)$ (by Lemma 8.2), the first rule to be applied is $\varphi_{\mathit{One}(S')}(f(x_1, \ldots, x_n)) \twoheadrightarrow \varphi_{f_1}(\varphi_{S'}(x_1), x_2, \ldots, x_n)$, which generates $\varphi_{f_1}(\varphi_{S'}(t_1), t_2, \ldots, t_n)$. Only $\mathbb{T}(S')$ (or $\mathbb{B}(\Gamma)$ if $S' = X$) can reduce $\varphi_{S'}(t_1)$. Suppose $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_1) \longrightarrow\!\!\!\!\!\rightarrow u$ with $u \in \mathcal{T}(\mathcal{F})$. Then we can only apply the rule $\varphi_{f_1}(x_1 \mathbin{@} !\bot(\_), x_2, \ldots, x_n) \twoheadrightarrow f(x_1, \ldots, x_n)$ to $\varphi_{f_1}(u, t_2, \ldots, t_n)$, and we obtain $f(u, t_2, \ldots, t_n)$, in contradiction with the initial hypothesis. Therefore, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_1) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_1')$, in less steps than the original reduction. By induction, $t_1' = t_1$. We can only apply the rule $\varphi_{f_1}(\bot(x_1), x_2, \ldots, x_n) \twoheadrightarrow \varphi_{f_2}(\bot(x_1), \varphi_S(x_2), \ldots, x_n)$ to $\varphi_{f_1}(\bot(t_1), t_2, \ldots, t_n)$, and we obtain $\varphi_{f_2}(\bot(t_1), \varphi_{S'}(t_2), \ldots, t_n)$. With the same reasoning, we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_i)$ for all i, and we therefore have $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mathit{One}(S')) \bullet \varphi_{S'}(\varphi_{f_1}(\varphi_{S'}(t_1), t_2, \ldots, t_n)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{f_n}(\bot(t_1), \ldots, \bot(t_n))$. We can apply only the rule $\varphi_{f_n}(\bot(x_1), \ldots, \bot(x_n)) \twoheadrightarrow \bot(f(x_1, \ldots, x_n))$ to the latter term, and we obtain $\bot(f(t_1, \ldots t_n))$, as required. ☐

Theorem 3.5 (Simulation). *Given a term $t \in \mathcal{T}(\mathcal{F})$, a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$,*
(1) $\Gamma \vdash S \circ t \Longrightarrow t'$ *iff* $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow t'$, $t' \in \mathcal{T}(\mathcal{F})$
(2) $\Gamma \vdash S \circ t \Longrightarrow \texttt{Fail}$ *iff* $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$

Proof. By induction on the height of derivation tree and respectively of on the length of the reduction.

$$S \;:=\; \mathit{Identity} \mid \mathit{Fail} \mid l \twoheadrightarrow r \mid S_1; S_2 \mid S_1 \leftrightarrow S_2 \mid \mathit{One}(s) \mid \mathit{All}(s) \mid \mu X.S'$$

Base case: $S := \mathit{Identity} \mid \mathit{Fail} \mid l \twoheadrightarrow r$

(1) $S := \mathit{Identity}$
Independently of Γ, for all $t \in \mathcal{T}(\mathcal{F})$ we have

$$\Gamma \vdash \mathit{Identity} \circ t \Longrightarrow t$$

and

$$\{\varphi_{\mathit{Identity}}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x\ \} \cup \mathbb{B}(\Gamma) \bullet \varphi_{\mathit{Identity}}(t) \longrightarrow\!\!\!\!\rightarrow t$$

(2) $S := \mathit{Fail}$
Independently of Γ, for all $t \in \mathcal{T}(\mathcal{F})$ we have

$$\Gamma \vdash \mathit{Fail} \circ t \Longrightarrow \texttt{Fail}$$

and

$$\{\varphi_{\mathit{Fail}}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \bot(x)\ \} \cup \mathbb{B}(\Gamma) \bullet \varphi_{\mathit{Fail}}(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$$

(3) $S := l \twoheadrightarrow r$

(a) $S \Rightarrow \mathbb{T}(S) \cup \mathbb{B}(\Gamma)$
If $\Gamma \vdash l \twoheadrightarrow r \circ t \Longrightarrow u$ then $\exists\sigma, \sigma(l) = t$ and $\sigma(r) = u$. Then, $\sigma(\varphi_{l\twoheadrightarrow r}(l)) = \varphi_{l\twoheadrightarrow r}(t)$ and thus $\varphi_{l\twoheadrightarrow r}(l) \twoheadrightarrow r \bullet \varphi_{l\twoheadrightarrow r}(t) \longrightarrow\!\!\!\!\rightarrow u$.
If $\Gamma \vdash l \twoheadrightarrow r \circ t \Longrightarrow \texttt{Fail}$ then $\nexists\sigma, \sigma(l) = t$ and thus $\nexists\sigma, \sigma(\varphi_{l\twoheadrightarrow r}(l)) = \varphi_{l\twoheadrightarrow r}(t)$. The rewrite rule $\varphi_{l\twoheadrightarrow r}(l) \twoheadrightarrow r$ cannot be applied to $\varphi_{l\twoheadrightarrow r}(t)$ at the head position. Since $\varphi_{l\twoheadrightarrow r}$ is a fresh symbol ($\nexists p$ s.t. $t(p) = \varphi_{l\twoheadrightarrow r}$) the rule cannot be applied to another position of $\varphi_{l\twoheadrightarrow r}(t)$. Since $\nexists\sigma, \sigma(\varphi_{l\twoheadrightarrow r}(l)) = \varphi_{l\twoheadrightarrow r}(t)$ then $\varphi_{l\twoheadrightarrow r}(t)$ is in the semantics of $\varphi_{l\twoheadrightarrow r}(!l)$ and thus the rule $\varphi_{l\twoheadrightarrow r}(x \mathbin{@} !l) \twoheadrightarrow \bot(x)$ can be applied and the result is $\bot(t)$.

(b) $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \Rightarrow S$
If the rule $\varphi_{l\twoheadrightarrow r}(l) \twoheadrightarrow r$ is used then, since $\varphi_{l\twoheadrightarrow r}$ does not occur in t, it can be only applied at the head position of $\varphi_{l\twoheadrightarrow r}(t)$ and thus $\exists\sigma, \sigma(\varphi_{l\twoheadrightarrow r}(l)) = \varphi_{l\twoheadrightarrow r}(t)$ and $\sigma(r) = u$. Since u contains no $\varphi_{l\twoheadrightarrow r}$ (the codomain of σ contains no $\varphi_{l\twoheadrightarrow r}$ because it does not occur in t) and the only rules in $\mathbb{B}(\Gamma)$ concerning $\varphi_{l\twoheadrightarrow r}$ could be at most the same as those in $\mathbb{T}(l \twoheadrightarrow r)$ then u is in normal form w.r.t. $\mathbb{T}(l \twoheadrightarrow r) \cup \mathbb{B}(\Gamma)$. We have then that $\exists\sigma, \sigma(l) = t$ and thus $l \twoheadrightarrow r \circ t \Longrightarrow u$. Since $\varphi_{l\twoheadrightarrow r}$ is a fresh symbol then there is no other way to apply the rewrite rule $\varphi_{l\twoheadrightarrow r}(l) \twoheadrightarrow r$ to $\varphi_{l\twoheadrightarrow r}(t)$.
If the rule $\varphi_{l\twoheadrightarrow r}(x \mathbin{@} !l) \twoheadrightarrow \bot(x)$ is applied then it is applied at the top position ($\varphi_{l\twoheadrightarrow r}$ fresh) and $\exists\sigma = \{t/x\}, \sigma(\varphi_{l\twoheadrightarrow r}(x \mathbin{@} !l)) = \varphi_{l\twoheadrightarrow r}(t)$ and $\nexists\mu, \mu(l) = \varphi_{l\twoheadrightarrow r}(t)$. Consequently, $\Gamma \vdash l \twoheadrightarrow r \circ t \Longrightarrow \texttt{Fail}$. We have $\varphi_{l\twoheadrightarrow r}(x \mathbin{@} !l) \twoheadrightarrow \bot(x) \bullet t \longrightarrow\!\!\!\!\rightarrow \bot(t)$, and because φ symbols do not occur in t, then $\bot(t)$ is in normal form w.r.t. $\mathbb{T}(l \twoheadrightarrow r) \cup \mathbb{B}(\Gamma)$. Because $\bot$ is a fresh symbol then $\nexists\sigma, \sigma(\bot(x)) = t$ and the rule $\varphi_{l\twoheadrightarrow r}(\bot(x)) \twoheadrightarrow \bot(x)$ cannot be applied to $\varphi_{l\twoheadrightarrow r}(t)$.

Induction: $S := S_1; S_2 \mid S_1 \leftarrow\!\!\!+ S_2 \mid \mathit{One}(S') \mid \mathit{All}(S') \mid \mu X.S'$

(1) $S := S_1; S_2$

(a) $S \Rightarrow \mathbb{T}(S)$
Since $\Gamma \vdash S_1; S_2 \circ t \Longrightarrow u$ then $\Gamma \vdash S_1 \circ t \Longrightarrow v$ and $\Gamma \vdash S_2 \circ v \Longrightarrow u$ with a shorter derivation tree. By induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\rightarrow v$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(v) \longrightarrow\!\!\!\!\rightarrow u$. We have $\varphi_{S_1;S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_;(\varphi_{S_2}(\varphi_{S_1}(x)), x) \bullet \varphi_{S_1;S_2}(t) \longrightarrow\!\!\!\!\rightarrow \varphi_;(\varphi_{S_2}(\varphi_{S_1}(t)), t)$. By the induction hypothesis, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_;(\varphi_{S_2}(\varphi_{S_1}(t)), t) \longrightarrow\!\!\!\!\rightarrow \varphi_;(\varphi_{S_2}(v), t)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_;(\varphi_{S_2}(v), t) \longrightarrow\!\!\!\!\rightarrow \varphi_;(u, t)$. Finally, $\varphi_;(x @ !\bot(\_), \_) \twoheadrightarrow x \bullet \varphi_;(u, t) \longrightarrow\!\!\!\!\rightarrow u$.

If $\Gamma \vdash S_1; S_2 \circ t \Longrightarrow \texttt{Fail}$ then $\Gamma \vdash S_1 \circ t \Longrightarrow \texttt{Fail}$, or $\Gamma \vdash S_1 \circ t \Longrightarrow v$ and $\Gamma \vdash S_2 \circ v \Longrightarrow \texttt{Fail}$. For the first case, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \twoheadrightarrow \bot(t)$. We have $\varphi_{S_1;S_2}(x \,@\, !\bot(\_)) \rightarrowtriangle \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x) \bullet \varphi_{S_1;S_2}(t) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t)$ and, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\bot(t)), t)$. By Lemma 8.1, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{;}(\varphi_{S_2}(\bot(t)), t) \twoheadrightarrow \varphi_{;}(\bot(t), t)$ and $\varphi_{;}(\bot(\_), x) \rightarrowtriangle \bot(x) \bullet \varphi_{;}(\bot(t), t) \twoheadrightarrow \bot(t)$. For the second case, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \twoheadrightarrow v$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(v) \twoheadrightarrow \bot(v)$. We have, $\varphi_{S_1;S_2}(x \,@\, !\bot(\_)) \rightarrowtriangle \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x) \bullet \varphi_{S_1;S_2}(t) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t)$. By induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(v), t)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{;}(\varphi_{S_2}(v), t) \twoheadrightarrow \varphi_{;}(\bot(v), t)$. Finally, $\varphi_{;}(\bot(\_), x) \rightarrowtriangle \bot(x) \bullet \varphi_{;}(\bot(v), t) \twoheadrightarrow \bot(t)$.

(b) $\mathbb{T}(S) \Rightarrow S$

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_1;S_2}(t) \twoheadrightarrow u$, since $t \in \mathcal{T}(\mathcal{F})$ contains no φ symbols, the first reduction is necessarily $\varphi_{S_1;S_2}(x \,@\, !\bot(\_)) \rightarrowtriangle \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x) \bullet \varphi_{S_1;S_2}(t) \longrightarrow \rightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t)$. According to Lemma 8.2, $\varphi_{S_1}(t)$ can only be reduced by $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1)$, which gives $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(v), t)$ for some v. If we had $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \twoheadrightarrow \bot(t)$, the $\bot$ would have been propagated (by Lemma 8.1) and the final result would have been $\bot(t)$ which contradicts the initial hypothesis. By induction (the latter reduction needs less steps than the initial one), $\Gamma \vdash S_1 \circ t \Longrightarrow v$. Next, the only possible reduction is $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{;}(\varphi_{S_2}(v), t) \twoheadrightarrow \varphi_{;}(u, t)$ and then $\varphi_{;}(x @ !\bot(\_), \_) \rightarrowtriangle x \bullet \varphi_{;}(u, t) \twoheadrightarrow u$. By induction, $\Gamma \vdash S_2 \circ v \Longrightarrow u$ and since $\Gamma \vdash S_1 \circ t \Longrightarrow v$ we can conclude that $\Gamma \vdash S_1; S_2 \circ t \Longrightarrow u$.

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_1;S_2}(t) \twoheadrightarrow \bot(t)$, since $\varphi_{S_1;S_2}(t)$ is in normal form w.r.t. $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2)$ (according to Lemma 8.2), the first reduction is necessarily $\varphi_{S_1;S_2}(x \,@\, !\bot(\_)) \rightarrowtriangle \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x) \bullet \varphi_{S_1;S_2}(t) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t)$. The only way to obtain $\bot(t)$ as a result is to have a reduction $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_2}(\varphi_{S_1}(t)) \twoheadrightarrow \bot(v)$. Thus, either $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_1}(t) \twoheadrightarrow v'$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_2}(v') \twoheadrightarrow \bot(v)$ or $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_1}(t) \twoheadrightarrow \bot(v)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1; S_2) \bullet \varphi_{S_2}(\bot(v)) \twoheadrightarrow \bot(v)$ (by Lemma 8.1). In the first case, we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \twoheadrightarrow v'$ since only the rules in $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1)$ can reduce $\varphi_{S_1}(t)$ and, by induction, $\Gamma \vdash S_1 \circ t \Longrightarrow v'$. Similarly, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(v') \twoheadrightarrow \bot(v)$ and, by Lemma 8.4, $v' = v$. Thus, by induction, $\Gamma \vdash S_2 \circ v' \Longrightarrow \texttt{Fail}$. Consequently, $\Gamma \vdash S_1; S_2 \circ t \Longrightarrow \texttt{Fail}$. For the second case, as previously, we have that $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \twoheadrightarrow \bot(v)$ and, by Lemma 8.4, $v = t$. By induction, $\Gamma \vdash S_1 \circ t \Longrightarrow \texttt{Fail}$ and consequently, $\Gamma \vdash S_1; S_2 \circ t \Longrightarrow \texttt{Fail}$.

(2) $S := \mu X.S'$

(a) $S \Rightarrow \mathbb{T}(S)$

Since $\Gamma \vdash \mu X.S' \circ t \Longrightarrow u$ then $\Gamma; X\colon S' \vdash S' \circ t \Longrightarrow u$ with the latter having a shorter derivation tree and thus, by induction, $\mathbb{B}(\Gamma; X\colon S') \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \twoheadrightarrow u$. The only possible reduction of $\varphi_{\mu X.S'}(t)$ *w.r.t.* $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S')$ is obtained by applying $\varphi_{\mu X.S'}(Y \,@\, !\bot(\_)) \rightarrowtriangle \varphi_{S'}(Y)$ which results in the term $\varphi_{S'}(t)$. Since, by induction, $\mathbb{B}(\Gamma; X\colon S') \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \twoheadrightarrow u$ and since $\mathbb{B}(\Gamma; X\colon S') \cup \mathbb{T}(S') \subseteq \mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S')$ we eventually have $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{\mu X.S'}(t) \twoheadrightarrow u$.

If $\Gamma \vdash \mu X.S' \circ t \Longrightarrow \texttt{Fail}$ then $\Gamma; X\colon S' \vdash S' \circ t \Longrightarrow \texttt{Fail}$. Using the same reasoning as for the successful case we obtain $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{\mu X.S'}(t) \twoheadrightarrow \bot(t)$.

(b) $\mathbb{T}(S) \Rightarrow S$

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{\mu X.S'}(t) \twoheadrightarrow u$ since $t \in \mathcal{T}(\mathcal{F})$ contains no φ symbols and $\mathbb{B}(\Gamma)$ contains no rules potentially rewriting $\varphi_{\mu X.S'}(t)$, the first reduction is necessarily

$\varphi_{\mu X.S'}(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y) \bullet \varphi_{\mu X.S'}(t) \longrightarrow\!\!\!\!\rightarrow \varphi_{S'}(t)$ and thus $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow u$ in strictly less steps than the original reduction. We also have that $\mathbb{B}(\Gamma; X : S') \cup \mathbb{T}(S') = \mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \setminus \{\varphi_{\mu X.S'}(\bot(Y)) \twoheadrightarrow \bot(Y), \varphi_{\mu X.S'}(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y)\}$ and since $\varphi_{S'}(t)$ and all its reducts *w.r.t.* $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S')$ contain no $\varphi_{\mu X.S'}$ then $\mathbb{B}(\Gamma; X : S') \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow u$ (in strictly less steps than the original reduction). By induction, $\Gamma; X : S' \vdash S' \circ t \Longrightarrow u$ and thus $\Gamma \vdash \mu X.S' \circ t \Longrightarrow u$.

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{\mu X.S'}(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$ the first reduction is still $\varphi_{\mu X.S'}(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y) \bullet \varphi_{\mu X.S'}(t) \longrightarrow\!\!\!\!\rightarrow \varphi_{S'}(t)$ and thus $\mathbb{B}(\Gamma) \cup \mathbb{T}(\mu X.S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow \bot(u)$ in strictly less steps than the original reduction. With the same arguments as before we obtain $\Gamma; X : S' \vdash S' \circ t \Longrightarrow \texttt{Fail}$ and thus $\Gamma \vdash \mu X.S' \circ t \Longrightarrow \texttt{Fail}$.

(3) $S := X$
  (a) $S \Rightarrow \mathbb{T}(S)$
  Since $\Gamma; X : S' \vdash X \circ t \Longrightarrow u$ then $\Gamma; X : S' \vdash S' \circ t \Longrightarrow u$ with the latter having a shorter derivation tree and thus, by induction, $\mathbb{B}(\Gamma; X : S') \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow u$. Since we supposed all bound variables (by a μ operator or a context assignment) to have different names, the only rewrite rules in $\mathbb{B}(\Gamma; X : S')$ involving φ_X are the ones generated by the context assignment: $\varphi_X(\bot(Y)) \twoheadrightarrow \bot(Y)$ and $\varphi_X(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y)$. Consequently, the only possible reduction of $\varphi_X(t)$ *w.r.t.* $\mathbb{B}(\Gamma; X : S')$ is obtained by applying the latter rule which results in the term $\varphi_{S'}(t)$. It is easy to check that $\mathbb{B}(\Gamma; X : S') \cup \mathbb{T}(S') = \mathbb{B}(\Gamma; X : S')$. Consequently $\mathbb{B}(\Gamma; X : S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow u$ and thus $\mathbb{B}(\Gamma; X : S') \bullet \varphi_X(t) \longrightarrow\!\!\!\!\rightarrow u$. If $\Gamma; X : S' \vdash X \circ t \Longrightarrow \texttt{Fail}$ then $\Gamma; X : S' \vdash S' \circ t \Longrightarrow \texttt{Fail}$. Using the same reasoning as for the successful case we obtain $\mathbb{B}(\Gamma; X : S') \bullet \varphi_X(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$.
  (b) $\mathbb{T}(S) \Rightarrow S$
  If $\mathbb{B}(\Gamma; X : S') \bullet \varphi_X(t) \longrightarrow\!\!\!\!\rightarrow u$ since $t \in \mathcal{T}(\mathcal{F})$ contains no φ symbols and since, as explained before the only rewrite rules in $\mathbb{B}(\Gamma; X : S')$ involving φ_X are the ones generated by the context assignment, the first reduction is necessarily $\varphi_X(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y) \bullet \varphi_X(t) \longrightarrow\!\!\!\!\rightarrow \varphi_{S'}(t)$ and thus $\mathbb{B}(\Gamma; X : S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow u$ in strictly less steps than the original reduction. We have $\mathbb{B}(\Gamma; X : S') \cup \mathbb{T}(S') = \mathbb{B}(\Gamma; X : S')$ and thus $\mathbb{B}(\Gamma; X : S') \cup \mathbb{T}(S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow u$ (in strictly less steps than the original reduction). By induction, $\Gamma; X : S' \vdash S' \circ t \Longrightarrow u$ and thus $\Gamma; X : S' \vdash X \circ t \Longrightarrow u$. If $\mathbb{B}(\Gamma; X : S') \bullet \varphi_X(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$ the first reduction is still $\varphi_X(Y \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{S'}(Y) \bullet \varphi_X(t) \longrightarrow\!\!\!\!\rightarrow \varphi_{S'}(t)$ and thus $\mathbb{B}(\Gamma; X : S') \bullet \varphi_{S'}(t) \longrightarrow\!\!\!\!\rightarrow \bot(u)$ in strictly less steps than the original reduction. With the same arguments as before we obtain $\Gamma; X : S' \vdash S' \circ t \Longrightarrow \texttt{Fail}$ and thus $\Gamma; X : S' \vdash X \circ t \Longrightarrow \texttt{Fail}$.

(4) $S := S_1 \nleftrightarrow S_2$
  (a) $S \Rightarrow \mathbb{T}(S)$
  If $S_1 \nleftrightarrow S_2 \circ t \Longrightarrow u$ then $S_1 \circ t \Longrightarrow u$ or, $S_1 \circ t \Longrightarrow \texttt{Fail}$ and $S_2 \circ t \Longrightarrow r$. In the first case, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\rightarrow v$. We have, $\varphi_{S_1 \nleftrightarrow S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\nleftrightarrow}(\varphi_{S_1}(x)) \bullet \varphi_{S_1 \nleftrightarrow S_2}(t) \longrightarrow\!\!\!\!\rightarrow \varphi_{\nleftrightarrow}(\varphi_{S_1}(t))$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}_{\varphi_1}(S_1) \bullet \varphi_{\nleftrightarrow}(\varphi_{S_1}(t)) \longrightarrow\!\!\!\!\rightarrow \varphi_{\nleftrightarrow}(u)$. Finally, $\varphi_{\nleftrightarrow}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x \bullet \varphi_{\nleftrightarrow}(u) \longrightarrow\!\!\!\!\rightarrow u$. For the second case, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(t) \longrightarrow\!\!\!\!\rightarrow u$. We have, as before, $\varphi_{S_1 \nleftrightarrow S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\nleftrightarrow}(\varphi_{S_1}(x)) \bullet \varphi_{S_1 \nleftrightarrow S_2}(t) \longrightarrow\!\!\!\!\rightarrow \varphi_{\nleftrightarrow}(\varphi_{S_1}(t))$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{\nleftrightarrow}(\varphi_{S_1}(t)) \longrightarrow\!\!\!\!\rightarrow \varphi_{\nleftrightarrow}(\bot(u))$. Then, $\varphi_{\nleftrightarrow}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x) \bullet \varphi_{\nleftrightarrow}(\bot(u)) \longrightarrow\!\!\!\!\rightarrow \varphi_{S_2}(u)$ and finally, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_2(t) \longrightarrow\!\!\!\!\rightarrow u$.

If $S_1 \leftarrow\!\!\!+ S_2 \circ t \Longrightarrow \texttt{Fail}$ then $S_1 \circ t \Longrightarrow \texttt{Fail}$ and $S_2 \circ t \Longrightarrow \texttt{Fail}$. By induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$. We have, $\varphi_{S_1 \leftarrow\!\!\!+ S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(x)) \bullet \varphi_{S_1 \leftarrow\!\!\!+ S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(t))$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(t)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{\leftarrow\!\!\!+}(\bot(u))$. Then, $\varphi_{\leftarrow\!\!\!+}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x) \bullet \varphi_{\leftarrow\!\!\!+}(\bot(u)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{S_2}(u)$ and finally, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_2(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(u)$.

(b) $\mathbb{T}(S) \Rightarrow S$

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1 \leftarrow\!\!\!+ S_2) \bullet \varphi_{S_1 \leftarrow\!\!\!+ S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow u$, since $t \in \mathcal{T}(\mathcal{F})$ contains no φ symbols, the first reduction is necessarily $\varphi_{S_1 \leftarrow\!\!\!+ S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(x)) \bullet \varphi_{S_1 \leftarrow\!\!\!+ S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(t))$. No rule can be applied at the top position until $\varphi_{S_1}(t)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$ or to a term of the form $\bot(\_)$. In the former case, we can only have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow u$ in less steps than the original reduction, so by induction, $S_1 \circ t \Longrightarrow u$. Consequently, $S_1 \leftarrow\!\!\!+ S_2 \circ t \Longrightarrow u$. In the latter case, we necessarily have (Lemma 8.4) $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(t)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{\leftarrow\!\!\!+}(\bot(t))$. Then we have $\varphi_{\leftarrow\!\!\!+}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x) \bullet \varphi_{\leftarrow\!\!\!+}(\bot(t)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{S_2}(t)$, which can only be reduced as $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow u$. By induction, $S_1 \circ t \Longrightarrow \texttt{Fail}$ and $S_2 \circ t \Longrightarrow u$ and consequently, $S_1 \leftarrow\!\!\!+ S_2 \circ t \Longrightarrow u$.

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1 \leftarrow\!\!\!+ S_2) \bullet \varphi_{S_1 \leftarrow\!\!\!+ S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$, since $\varphi_{S_1 \leftarrow\!\!\!+ S_2}(t)$ is in normal form w.r.t. $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1)$ and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2)$ (Lemma 8.2), the first reduction is necessarily $\varphi_{S_1 \leftarrow\!\!\!+ S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(x)) \bullet \varphi_{S_1 \leftarrow\!\!\!+ S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(t))$. No rule can be applied at the top position until $\varphi_{S_1}(t)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$ or to a term of the form $\bot(\_)$. If it is a term in $\mathcal{T}(\mathcal{F})$ then the rule $\varphi_{\leftarrow\!\!\!+}(x @ !\bot(\_)) \twoheadrightarrow x$ is the only one that can be applied and the final result is not $\bot(t)$. Thus, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{S_1}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$ (in less steps than the original reduction) and $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_1) \bullet \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(t)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{\leftarrow\!\!\!+}(\bot(t))$. The only possible reduction is $\varphi_{\leftarrow\!\!\!+}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x) \bullet \varphi_{\leftarrow\!\!\!+}(\bot(t)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{S_2}(t)$ and then $\mathbb{B}(\Gamma) \cup \mathbb{T}(S_2) \bullet \varphi_{S_2}(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$ in less steps than the original reduction. By induction, $S_1 \circ t \Longrightarrow \texttt{Fail}$ and $S_2 \circ t \Longrightarrow \texttt{Fail}$ and consequently, $S_1 \leftarrow\!\!\!+ S_2 \circ t \Longrightarrow \texttt{Fail}$.

(5) $S := \mathit{All}(S')$

(a) $S \Rightarrow \mathbb{T}(S)$

If $\mathit{All}(S') \circ t \Longrightarrow u$ with $t = f(t_1, \ldots, t_n)$ then $\forall i \in [1,n], S' \circ t_i \Longrightarrow u_i$ and $u = f(u_1, \ldots, u_n)$. Then, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow u_i$. When we apply the rules corresponding to the encoding of All we obtain

$$\varphi_{\mathit{All}(S')}(f(x_1, \ldots, x_n)) \twoheadrightarrow \varphi_f(\varphi_{S'}(x_1), \ldots, \varphi_{S'}(x_n), f(x_1, \ldots, x_n))$$
$$\bullet \varphi_{\mathit{All}(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_f(\varphi_{S'}(t_1), \ldots, \varphi_{S'}(t_n), f(t_1, \ldots, t_n))$$

and no other rule can be applied at the top position for this latter term. By the induction hypothesis we have $\varphi_f(\varphi_{S'}(t_1), \ldots, \varphi_{S'}(t_n), f(t_1, \ldots, t_n)) \xrightarrow{\mathbb{B}(\Gamma) \cup \mathbb{T}(S')^*} \varphi_f(u_1, \ldots, u_n, f(t_1, \ldots, t_n))$ and subsequently

$$\varphi_f(x_1 \mathbin{@} !\bot(\_), \ldots, x_n \mathbin{@} !\bot(\_), \_) \twoheadrightarrow f(x_1, \ldots, x_n)$$
$$\bullet \varphi_f(u_1, \ldots, u_n, f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\rightarrow f(u_1, \ldots, u_n)$$

If $\mathit{All}(S') \circ t \Longrightarrow \texttt{Fail}$ with $t = f(t_1, \ldots, t_n)$ then $\exists i \in [1,n], S' \circ t_i \Longrightarrow \texttt{Fail}$. By induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_i)$. As before, when we apply the rules corresponding to the encoding of All we obtain

$$\varphi_{All(S')}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_f(\varphi_{S'}(x_1),\ldots,\varphi_{S'}(x_n),f(x_1,\ldots,x_n)) \\ \bullet\varphi_{All(S')}(f(t_1,\ldots,t_n)) \longrightarrow\!\!\!\!\rightarrow \varphi_f(\varphi_{S'}(t_1),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n))$$

and using induction

$$\varphi_f(\varphi_{S'}(t_1),\ldots,\varphi_{S'}(t_i),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n)) \\ \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')}{\longrightarrow\!\!\!\!\rightarrow} \varphi_f(u_1,\ldots,\bot(t_i),\ldots,u_n,f(t_1,\ldots,t_n)).$$

We have then

$$\varphi_f(\bot(\_),\ldots,\_,z) \twoheadrightarrow \bot(z) \\ \bullet\varphi_f(u_1,\ldots,\bot(t_i),\ldots,u_n,f(t_1,\ldots,t_n)) \longrightarrow\!\!\!\!\rightarrow \bot(f(t_1,\ldots,t_n))$$

as required.

If t is a constant c then $All(S')\circ c \Longrightarrow c$. We have $\varphi_{All(S')}(c) \twoheadrightarrow \varphi_c(c)\bullet\varphi_{All(S')}(c) \longrightarrow\!\!\!\!\rightarrow \varphi_c(c)$ and $\varphi_c(\_) \twoheadrightarrow c \bullet \varphi_c(c) \longrightarrow\!\!\!\!\rightarrow c$.

(b) $\mathbb{T}(S) \Rightarrow S$

We consider $t = f(t_1,\ldots,t_n)$ and handle the case of a constant later on. If $\mathbb{B}(\Gamma) \cup \mathbb{T}(All(S')) \bullet \varphi_{All(S')}(t) \longrightarrow\!\!\!\!\rightarrow u$, since $t \in \mathcal{T}(\mathcal{F})$ contains no φ symbols, we have first $\varphi_{All(S')}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_f(\varphi_{S'}(x_1),\ldots,\varphi_{S'}(x_n),f(x_1,\ldots,x_n)) \bullet \varphi_{All(S')}(f(t_1,\ldots,t_n)) \longrightarrow\!\!\!\!\rightarrow \varphi_f(\varphi_{S'}(t_1),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n))$. No rule can be applied at the top position until the $\varphi_{S'}(t_i)$ are reduced to terms in $\mathcal{T}(\mathcal{F})$ or at least one of them is reduced to a term of the form $\bot(\_)$. The former case holds only if $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\rightarrow u_i$ (in less steps than the original reduction) and

$$\varphi_f(\varphi_{S'}(t_1),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n)) \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')^*}{\longrightarrow\!\!\!\!\rightarrow} \varphi_f(u_1,\ldots,u_n,f(t_1,\ldots,t_n)).$$

This term can be further reduced only by the rule $\varphi_f(x_1 @ ! \bot(\_),\ldots,x_n @ ! \bot(\_),\_) \twoheadrightarrow f(x_1,\ldots,x_n)$ to $f(u_1,\ldots,u_n) = u$. By induction, $S' \circ t_i \Longrightarrow u_i$. Consequently, $All(S')\circ t \Longrightarrow u$. In the latter case, we necessarily have (Lemmas 8.4) $\mathbb{B}(\Gamma)\cup\mathbb{T}(S')\bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\rightarrow \bot(t_i)$ for some i, and then

$$\varphi_f(\ldots,\bot(\_),\ldots,\_,z) \twoheadrightarrow \bot(z) \\ \bullet\varphi_f(\varphi_{S'}(t_1),\ldots,\bot(t_i),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n)) \longrightarrow\!\!\!\!\rightarrow \bot(f(t_1,\ldots,t_n))$$

We obtain a term not in $\mathcal{T}(\mathcal{F})$, hence a contradiction with the original hypothesis. If $\mathbb{B}(\Gamma) \cup \mathbb{T}(All(S')) \bullet \varphi_{All(S')}(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$, since $\varphi_{All(S')}(t)$ is in normal form w.r.t. $\mathbb{B}(\Gamma) \cup \mathbb{T}(S')$ (Lemma 8.2), the first reduction can only be, as before,

$$\varphi_{All(S')}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_f(\varphi_{S'}(x_1),\ldots,\varphi_{S'}(x_n),f(x_1,\ldots,x_n)) \\ \bullet\varphi_{All(S')}(f(t_1,\ldots,t_n)) \longrightarrow\!\!\!\!\rightarrow \varphi_f(\varphi_{S'}(t_1),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n))$$

and no rule can be applied at the top position until the $\varphi_{S'}(t_i)$ are reduced to terms in $\mathcal{T}(\mathcal{F})$ or at least one of them is reduced to a term of the form $\bot(\_)$. We already handled the former case just before. As we have seen, for the latter case we have $\mathbb{B}(\Gamma)\cup\mathbb{T}(S')\bullet\varphi_{S'}(t_i) \longrightarrow\!\!\!\!\rightarrow \bot(t_i)$ for some i in less steps than the original reduction, and eventually $\varphi_f(\ldots,\bot(\_),\ldots,\_,z) \twoheadrightarrow \bot(z) \bullet \varphi_f(\varphi_{S'}(t_1),\ldots,\varphi_{S'}(t_n),f(t_1,\ldots,t_n)) \longrightarrow\!\!\!\!\rightarrow \bot(f(t_1,\ldots,t_n))$. By induction, $S' \circ t_i \Longrightarrow \texttt{Fail}$ and thus $All(S') \circ f(t_1,\ldots,t_n) \Longrightarrow \texttt{Fail}$.

If t is a constant c then $\varphi_{All(S')}(c) \twoheadrightarrow \varphi_c(c) \bullet \varphi_{All(S')}(c) \longrightarrow\!\!\!\!\!\twoheadrightarrow \varphi_c(c)$ and $\varphi_c(\_) \twoheadrightarrow c \bullet \varphi_c(c) \longrightarrow\!\!\!\!\!\twoheadrightarrow c$. On the other hand we have $All(S') \circ c \Longrightarrow c$ which confirms the property.

(6) $S := One(S')$

(a) $S \Rightarrow \mathbb{T}(S)$

If $One(S') \circ t \Longrightarrow u$ with $t = f(t_1, \ldots, t_n)$ then $\exists i \in [1,n], S \circ t_i \Longrightarrow u_i$ and $u = f(t_1, \ldots, u_i \ldots, t_n)$. Then, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\twoheadrightarrow u_i$. The encoding implements a leftmost behaviour for *One*, *i.e.* it supposes that $\forall j < i, S' \circ t_j \Longrightarrow$ `Fail` and if we suppose that this assumption holds in our case then, by induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_j) \longrightarrow\!\!\!\!\!\twoheadrightarrow \bot(t_j)$ for all $j < i$. When we apply the rules corresponding to the encoding of *One* we obtain $\varphi_{One(S')}(f(x_1, \ldots, x_n)) \twoheadrightarrow \varphi_{f_1}(\varphi_{S'}(x_1), x_2, \ldots, x_n) \bullet \varphi_{One(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\twoheadrightarrow \varphi_{f_1}(\varphi_{S'}(t_1), t_2, \ldots, t_n)$. and no other rule can be applied at the top position for this latter term until $\varphi_{S'}(t_1)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$ or to a term of the form $\bot(\_)$. We have

$$\varphi_{f_1}(\varphi_{S'}(t_1), \ldots, t_n) \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} \varphi_{f_1}(\bot(t_1), \ldots, t_n)$$

and then

$$\varphi_{f_1}(\bot(x_1), x_2, \ldots, x_n) \twoheadrightarrow \varphi_{f_2}(\bot(x_1), \varphi_{S'}(x_2), \ldots, x_n) \bullet \varphi_{f_1}(\bot(t_1), \ldots, t_n) \longrightarrow\!\!\!\!\!\twoheadrightarrow \varphi_{f_2}(\bot(t_1), \varphi_{S'}(t_2) \ldots, t_n)\,.$$

Repeating the reasoning for all $j < i$, we eventually get

$$\varphi_{f_1}(\varphi_{S'}(t_1), \ldots, t_n) \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')^*}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} \varphi_{f_i}(\bot(t_1), \ldots, \bot(t_{i-1}), u_i \ldots, t_n)$$

and then

$$\varphi_{f_i}(\bot(x_1), \ldots, \bot(x_{i-1}), x_i\,@\,!\bot(\_), x_{i+1}, \ldots, x_n) \twoheadrightarrow f(x_1, \ldots, x_n) \bullet \varphi_{f_i}(\bot(t_1), \ldots, \bot(t_{i-1}), u_i \ldots, t_n) \longrightarrow\!\!\!\!\!\twoheadrightarrow \varphi_{f_i}(t_1, \ldots, u_i, \ldots, t_n)$$

as required.

If $One(S') \circ t \Longrightarrow$ `Fail` with $t = f(t_1, \ldots, t_n)$ then $\forall i \in [1,n], S' \circ t_i \Longrightarrow$ `Fail`. By induction, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\twoheadrightarrow \bot(t_i)$. As before, applying the rules corresponding to the encoding of *One* gives $\varphi_{One(S')}(f(x_1, \ldots, x_n)) \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')^*}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} \varphi_{f_n}(\bot(t_1), \ldots, \bot(t_n))$ and then

$$\varphi_{f_n}(\bot(x_1), \ldots, \bot(x_n)) \twoheadrightarrow \bot(f(x_1, \ldots, x_n)) \bullet \varphi_{f_n}(\bot(t_1), \ldots, \bot(t_n)) \longrightarrow\!\!\!\!\!\twoheadrightarrow \bot(f(t_1, \ldots, t_n))$$

as wished.

If t is a constant c then $One(S') \circ c \Longrightarrow$ `Fail`, and we have $\varphi_{One(S')}(c) \twoheadrightarrow \bot(c) \bullet \varphi_{(One(S'))}(c) \longrightarrow\!\!\!\!\!\twoheadrightarrow \bot(c)$ as well.

(b) $\mathbb{T}(S) \Rightarrow S$

If $\mathbb{B}(\Gamma) \cup \mathbb{T}(One(S')) \bullet \varphi_{One(S')}(t) \longrightarrow\!\!\!\!\!\twoheadrightarrow u$, then $t = f(t_1, \ldots, t_n)$, because if t is a constant c then we would necessarily have $\varphi_{One(S')}(c) \twoheadrightarrow \bot(c) \bullet \varphi_{One(S')}(c) \longrightarrow\!\!\!\!\!\twoheadrightarrow \bot(c)$, which contradicts the hypothesis. Because $t \in \mathcal{T}(\mathcal{F})$ contains no φ symbols, the first reduction is necessarily

$$\varphi_{One(S')}(f(x_1, \ldots, x_n)) \twoheadrightarrow \varphi_{f_1}(\varphi_{S'}(x_1), x_2, \ldots, x_n) \bullet \varphi_{One(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\twoheadrightarrow \varphi_{f_1}(\varphi_{S'}(t_1), t_2, \ldots, t_n)\,.$$

No rule can be applied at the top position until the $\varphi_{S'}(t_1)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$ or to a term of the form $\perp(\_)$. In the former case, we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_1) \longrightarrow\!\!\!\!\!\rightarrow u_1$ in less steps than the original reduction, and also $\varphi_{f_1}(\varphi_{S'}(t_1), \ldots, t_n) \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')}{\longrightarrow\!\!\!\!\!\rightarrow} \varphi_{f_1}(u_1, \ldots, t_n)$ which can be further reduced only by the rule $\varphi_{f_1}(x_1 \,@\, !\perp(\_), x_2, \ldots, x_n) \twoheadrightarrow f(x_1, \ldots, x_n)$ to $f(u_1, \ldots, t_n) = u$. By induction, $S' \circ t_1 \Longrightarrow u_1$, and consequently, $One(S') \circ t \Longrightarrow u$. In the latter case, we necessarily have (Lemma 8.4) $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_1) \longrightarrow\!\!\!\!\!\rightarrow \perp(t_1)$ and $\varphi_{f_1}(\varphi_{S'}(t_1), \ldots, t_n) \overset{\mathbb{B}(\Gamma)\cup\mathbb{T}(S')}{\longrightarrow\!\!\!\!\!\rightarrow} \varphi_{f_1}(\perp(t_1), \ldots, t_n)$ which can be further reduced only by $\varphi_{f_1}(\perp(x_1), x_2, \ldots, x_n) \twoheadrightarrow \varphi_{f_2}(\perp(x_1), \varphi_{S'}(x_2), \ldots, x_n)$ to $f(\perp(t_1), \varphi_{S'}(t_2), \ldots, t_n)$. Once again, no rule can be applied at the top position until the $\varphi_{S'}(t_2)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$, in which case we conclude as before, or to a term of the form $\perp(\_)$, in which case we continue the same way and we eventually get a term of the form $\varphi_{f_i}(\perp(t_1), \ldots, \perp(t_{i-1}), u_i, t_{i+1}, \ldots, t_n)$, then reduced by

$$\varphi_{f_i}(\perp(x_1), \ldots, \perp(x_{i-1}), x_i \,@\, !\perp(\_), x_{i+1}, \ldots, x_n) \twoheadrightarrow f(x_1, \ldots, x_n)$$

to $f(t_1, \ldots, u_i, t_{i+1}, \ldots, t_n) = u$. In this case we have, by induction, $S' \circ t_j \Longrightarrow \perp(t_j)$ for all $j < i$ and $S' \circ t_i \Longrightarrow u_i$, and thus, $One(S') \circ t \Longrightarrow u$. Note that we can always get an $u_i \in \mathcal{T}(\mathcal{F})$ at some point since otherwise we would eventually obtain the term $\varphi_{f_n}(\perp(t_1), \ldots, \perp(t_n))$ which can be reduce only to $\perp(f(t_1, \ldots, t_n))$ which is not a term in $\mathcal{T}(\mathcal{F})$ and thus contradicts the original hypothesis.
If $\mathbb{B}(\Gamma) \cup \mathbb{T}(One(S')) \bullet \varphi_{One(S')}(t) \longrightarrow\!\!\!\!\!\rightarrow \perp(t)$ and t is not a constant, because $\varphi_{One(S')}(t)$ is in normal form w.r.t. $\mathbb{B}(\Gamma) \cup \mathbb{T}(S')$ (Lemma 8.2), the first reduction can only be

$$\varphi_{One(S')}(f(x_1, \ldots, x_n)) \twoheadrightarrow \varphi_{f_1}(\varphi_{S'}(x_1), x_2, \ldots, x_n)$$
$$\bullet \varphi_{One(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\rightarrow \varphi_{f_1}(\varphi_{S'}(t_1), t_2, \ldots, t_n)$$

and no rule can be applied at the top position until the $\varphi_{S'}(t_1)$ is reduced to a term in $\mathcal{T}(\mathcal{F})$ or to a term of the form $\perp(\_)$. As we have seen just before, the former case leads to a term in $\mathcal{T}(\mathcal{F})$ which does not correspond to our hypothesis. We have also already handled the second case but we supposed that at some point we have $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow u_i$ which would lead to an eventual reduction to a term in $\mathcal{T}(\mathcal{F})$ which, once again, does not correspond to our hypothesis. The only remaining possibility is $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow \perp(\_)$ for all $i \leq n$ and thus, by Lemma 8.4, $\mathbb{B}(\Gamma) \cup \mathbb{T}(S') \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow \perp(t_i)$ (in less steps than the original reduction) for all $i \leq n$. In this case, we obtain $\varphi_{f_n}(\perp(x_1), \ldots, \perp(x_n)) \twoheadrightarrow \perp(f(x_1, \ldots, x_n)) \bullet \varphi_{f_n}(\perp(t_1), \ldots, \perp(t_n)) \longrightarrow\!\!\!\!\!\rightarrow \perp(f(t_1, \ldots, t_n))$. By induction, $S' \circ t_i \Longrightarrow \texttt{Fail}$ and thus $One(S') \circ f(t_1, \ldots, t_n) \Longrightarrow \texttt{Fail}$.
If t is a constant c then $\varphi_{One(S')}(c) \twoheadrightarrow \perp(c) \bullet \varphi_{One(S')}(c) \longrightarrow\!\!\!\!\!\rightarrow \perp(c)$. We also have $One(S') \circ c \Longrightarrow \texttt{Fail}$. □

Lemma 3.8. *Given a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, the TRS $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ is confluent.*

Proof. We consider two subsets of $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$: a set T_1 consisting of the rewrite rules in $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ obtained by expanding the rule schemas of the form $\varphi_{l \twoheadrightarrow r}(x \,@\, !l) \twoheadrightarrow \perp(x)$ and a set T_2 consisting of the remaining rules in $\mathbb{T}(S) \cup \mathbb{B}(\Gamma)$. Keeping in mind that all φ symbols are freshly generated we can easily notice that T_2 is orthogonal and thus confluent.

T_1 is linear but not orthogonal because of possible critical pairs between rules generated for a rewrite rule $l \twoheadrightarrow r$ by $\varphi_{l \twoheadrightarrow r}(x \mathbin{@} !l) \twoheadrightarrow \bot(x)$; there are no critical pairs between the rules generated for different rewrite rules because of the different φ head symbol in the left-hand sides of the corresponding rules. All these critical pairs are trivially joinable and since T_1 is also obviously terminating then it is also confluent. Since T_1 and T_2 are orthogonal to each other, *i.e.* there is no overlap between a rule from T_1 and one from T_2, then $T_1 \cup T_2 = \mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ is confluent [32]. □

9. Proofs for the Meta-Encoding

Lemma 9.1. *Given terms $t_1 \ldots t_n \in \mathcal{T}(\mathcal{F})$, S, and Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\rightarrow \ulcorner t_i' \urcorner$ with $\ulcorner t_i' \urcorner \neq \bot(\ulcorner t_i \urcorner)$ for all i iff*

$$\begin{aligned}&\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)\\ &\quad .\ \ \bullet \varphi'_{All(S)}(\varphi_S(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_1 \urcorner :: \mathtt{nil}, \mathtt{nil}) \longrightarrow\!\!\!\!\rightarrow \ulcorner t_1' \urcorner :: \ldots :: \ulcorner t_n' \urcorner :: \mathtt{nil}\end{aligned}$$

Proof. Because $\mathbb{T}_{\mathcal{M}}(All(S))$ contains $\mathbb{T}_{\mathcal{M}}(S)$, and by definition of $\mathbb{T}_{\mathcal{M}}(All(S))$, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\rightarrow \ulcorner t_i' \urcorner$ with $\ulcorner t_i' \urcorner \neq \bot(\ulcorner t_i \urcorner)$ for all i iff

$$\begin{aligned}&\varphi'_{All(S)}(\varphi_S(\ulcorner t_i \urcorner), \ulcorner t_{i+1} \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_i \urcorner :: \ldots \ulcorner t_1 \urcorner :: \mathtt{nil}, \ulcorner t_{i-1}' \urcorner :: \ldots :: \ulcorner t_1' \urcorner :: \mathtt{nil})\\ &\qquad\qquad\qquad\qquad \longrightarrow\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\\ &\quad \varphi'_{All(S)}(\varphi_S(\ulcorner t_{i+1} \urcorner), \ulcorner t_{i+2} \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_{i+1} \urcorner :: \ldots \ulcorner t_1 \urcorner :: \mathtt{nil}, \ulcorner t_i' \urcorner :: \ldots :: \ulcorner t_1' \urcorner :: \mathtt{nil})\end{aligned}$$

for all i. As a result, we deduce

$$\begin{aligned}&\qquad \varphi'_{All(S)}(\varphi_S(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_1 \urcorner :: \mathtt{nil}, \mathtt{nil})\\ \longrightarrow\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\varphi'_{All(S)}(\varphi_S(\ulcorner t_n \urcorner), \mathtt{nil}, \ulcorner t_n \urcorner :: \ldots \ulcorner t_1 \urcorner :: \mathtt{nil}, \ulcorner t_{n-1}' \urcorner :: \ldots :: \ulcorner t_1' \urcorner :: \mathtt{nil})\\ \longrightarrow\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\varphi'_{All(S)}(\ulcorner t_n' \urcorner, \mathtt{nil}, \ulcorner t_n \urcorner :: \ldots \ulcorner t_1 \urcorner :: \mathtt{nil}, \ulcorner t_{n-1}' \urcorner :: \ldots :: \ulcorner t_1' \urcorner :: \mathtt{nil})\\ \longrightarrow\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\mathtt{rev}(\ulcorner t_n' \urcorner :: \ulcorner t_{n-1}' \urcorner :: \ldots :: \ulcorner t_1' \urcorner :: \mathtt{nil})\\ \longrightarrow\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\ulcorner t_1' \urcorner :: \ldots :: \ulcorner t_n' \urcorner :: \mathtt{nil}\end{aligned}$$

as wished. □

Lemma 9.2. *Given terms $t_1 \ldots t_n \in \mathcal{T}(\mathcal{F})$, S, and Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_j \urcorner) \longrightarrow\!\!\!\!\rightarrow \ulcorner t_j' \urcorner$ with $\ulcorner t_j' \urcorner \neq \bot(\ulcorner t_j \urcorner)$ for all $j < i_0$ and $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_{i_0} \urcorner) \longrightarrow\!\!\!\!\rightarrow \bot(\ulcorner t_{i_0} \urcorner)$ iff*

$$\begin{aligned}&\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)\\ &.\ \ \bullet \varphi'_{All(S)}(\varphi_S(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_1 \urcorner :: \mathtt{nil}, \mathtt{nil}) \longrightarrow\!\!\!\!\rightarrow \bot_{list}(\ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})\end{aligned}$$

Proof. As in the proof of Lemma 9.1, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_j \urcorner) \longrightarrow\!\!\!\!\rightarrow \ulcorner t_j' \urcorner$ with $\ulcorner t_j' \urcorner \neq \bot(\ulcorner t_j \urcorner)$ for all $j < i_0$ iff

$$\begin{aligned}&\varphi'_{All(S)}(\varphi_S(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_1 \urcorner :: \mathtt{nil}, \mathtt{nil})\\ &\qquad\qquad\qquad\qquad \longrightarrow\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(All(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\\ &\quad \varphi'_{All(S)}(\varphi_S(\ulcorner t_{i_0} \urcorner), \ulcorner t_{i_0+1} \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}, \ulcorner t_{i_0} \urcorner :: \ldots \ulcorner t_1 \urcorner :: \mathtt{nil}, \ulcorner t_{i_0-1}' \urcorner :: \ldots :: \ulcorner t_1' \urcorner :: \mathtt{nil})\end{aligned}$$

$$
\begin{array}{lrcl}
\textbf{(ME1)} & \mathbb{T}_{\mathcal{M}}(\mathit{Identity}) & = & \{\ \varphi_{\mathit{Identity}}(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow x, \quad \varphi_{\mathit{Identity}}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\textbf{(ME2)} & \mathbb{T}_{\mathcal{M}}(\mathit{Fail}) & = & \{\ \varphi_{\mathit{Fail}}(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow \bot(x), \quad \varphi_{\mathit{Fail}}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\textbf{(ME3)} & \mathbb{T}_{\mathcal{M}}(l \twoheadrightarrow r) & = & \{\ \varphi_{l\twoheadrightarrow r}(\ulcorner l \urcorner) \twoheadrightarrow \ulcorner r \urcorner, \quad \varphi_{l\twoheadrightarrow r}(x \mathbin{@} \ulcorner ! l \urcorner) \twoheadrightarrow \bot(x), \\
 & & & \quad \varphi_{l\twoheadrightarrow r}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\textbf{(ME4)} & \mathbb{T}_{\mathcal{M}}(S_1;S_2) & = & \mathbb{T}_{\mathcal{M}}(S_1) \ \cup\ \mathbb{T}_{\mathcal{M}}(S_2) \\
 & & \bigcup & \{\ \varphi_{S_1;S_2}(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x), \\
 & & & \quad \varphi_{S_1;S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \quad \varphi_{;}(x \mathbin{@} \mathtt{appl}(\_,\_),\_) \twoheadrightarrow x, \quad \varphi_{;}(\bot(\_),x) \twoheadrightarrow \bot(x)\ \} \\
\textbf{(ME5)} & \mathbb{T}_{\mathcal{M}}(S_1 \leftarrow\!\!\!+ S_2) & = & \mathbb{T}_{\mathcal{M}}(S_1) \ \cup\ \mathbb{T}_{\mathcal{M}}(S_2) \\
 & & \bigcup & \{\ \varphi_{S_1 \leftarrow\!+ S_2}(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_{\leftarrow\!+}(\varphi_{S_1}(x)), \\
 & & & \quad \varphi_{S_1 \leftarrow\!+ S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \quad \varphi_{\leftarrow\!+}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x), \quad \varphi_{\leftarrow\!+}(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow x\ \} \\
\textbf{(ME6)} & \mathbb{T}_{\mathcal{M}}(\mu X\,.\,S) & = & \mathbb{T}_{\mathcal{M}}(S) \\
 & & \bigcup & \{\ \varphi_{\mu X.S}(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_S(x), \quad \varphi_{\mu X.S}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \quad \varphi_X(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_S(x), \quad \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\textbf{(ME7)} & \mathbb{T}_{\mathcal{M}}(X) & = & \emptyset \\
 & & & \\
\textbf{(ME8)} & \mathbb{T}_{\mathcal{M}}(\mathit{All}(S)) & = & \mathbb{T}_{\mathcal{M}}(S) \ \cup\ \mathbb{L} \\
 & & \bigcup & \{\ \varphi_{\mathit{All}(S)}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \quad \varphi_{\mathit{All}(S)}(\mathtt{appl}(f,\mathit{args})) \twoheadrightarrow \mathtt{propag}(\mathtt{appl}(f,\varphi^{\mathit{list}}_{\mathit{All}(S)}(\mathit{args}))), \\
 & & & \quad \varphi^{\mathit{list}}_{\mathit{All}(S)}(h :: q) \twoheadrightarrow \varphi'_{\mathit{All}(S)}(\varphi_S(h), q, h :: \mathtt{nil}, \mathtt{nil}), \\
 & & & \quad \varphi^{\mathit{list}}_{\mathit{All}(S)}(\mathtt{nil}) \twoheadrightarrow \mathtt{nil}, \\
 & & & \quad \varphi'_{\mathit{All}(S)}(\bot(\_), \mathit{todo}, \mathit{r\_tried}, \_) \twoheadrightarrow \bot_{\mathit{list}}(\mathtt{rconcat}(\mathit{r\_tried}, \mathit{todo})), \\
 & & & \quad \varphi'_{\mathit{All}(S)}(x \mathbin{@} \mathtt{appl}(\_,\_), \mathtt{nil}, \_, \mathit{r\_done}) \twoheadrightarrow \mathtt{rev}(x :: \mathit{r\_done}), \\
 & & & \quad \varphi'_{\mathit{All}(S)}(x \mathbin{@} \mathtt{appl}(\_,\_), h :: q, \mathit{r\_tried}, \mathit{r\_done}) \twoheadrightarrow \\
 & & & \qquad\qquad \varphi'_{\mathit{All}(S)}(\varphi_S(h), q, h :: \mathit{r\_tried}, x :: \mathit{r\_done})\ \} \\
 & & & \\
\textbf{(ME9)} & \mathbb{T}_{\mathcal{M}}(\mathit{One}(S)) & = & \mathbb{T}_{\mathcal{M}}(S) \ \cup\ \mathbb{L} \\
 & & \bigcup & \{\ \varphi_{\mathit{One}(S)}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \quad \varphi_{\mathit{One}(S)}(\mathtt{appl}(f,\mathit{args})) \twoheadrightarrow \mathtt{propag}(\mathtt{appl}(f,\varphi^{\mathit{list}}_{\mathit{One}(S)}(\mathit{args}))), \\
 & & & \quad \varphi^{\mathit{list}}_{\mathit{One}(S)}(h :: q) \twoheadrightarrow \varphi'_{\mathit{One}(S)}(\varphi_S(h), q, h :: \mathtt{nil}), \\
 & & & \quad \varphi^{\mathit{list}}_{\mathit{One}(S)}(\mathtt{nil}) \twoheadrightarrow \bot_{\mathit{list}}(\mathtt{nil}), \\
 & & & \quad \varphi'_{\mathit{One}(S)}(\bot(\_), \mathtt{nil}, \mathit{r\_tried}) \twoheadrightarrow \bot_{\mathit{list}}(\mathtt{rev}(\mathit{r\_tried})), \\
 & & & \quad \varphi'_{\mathit{One}(S)}(\bot(\_), h :: q, \mathit{r\_tried}) \twoheadrightarrow \varphi'_{\mathit{One}(S)}(\varphi_S(h), q, h :: \mathit{r\_tried}), \\
 & & & \quad \varphi'_{\mathit{One}(S)}(x \mathbin{@} \mathtt{appl}(\_,\_), \mathit{todo}, \_ :: \mathit{r\_tried}) \twoheadrightarrow \\
 & & & \qquad\qquad \mathtt{rconcat}(\mathit{r\_tried}, x :: \mathit{todo})\ \} \\
 & & & \\
\textbf{(ME10)} & \mathbb{B}_{\mathcal{M}}(\Gamma; X\colon S) & = & \mathbb{B}_{\mathcal{M}}(\Gamma) \ \cup\ \mathbb{T}_{\mathcal{M}}(S) \\
 & & \bigcup & \{\ \varphi_X(x \mathbin{@} \mathtt{appl}(\_,\_)) \twoheadrightarrow \varphi_S(x), \quad \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
\textbf{(ME11)} & \mathbb{B}_{\mathcal{M}}(\oslash) & = & \emptyset
\end{array}
$$

Figure 6: Strategy translation for meta-encoded terms; $\mathbb{L}$ is defined in Figure 4.

This latter term can only reduce to

$$\varphi'_{All(S)}(\bot\ulcorner t_{i_0}\urcorner, \ulcorner t_{i_0+1}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_{i_0}\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil}, \ulcorner t'_{i_0-1}\urcorner :: \ldots :: \ulcorner t'_1\urcorner :: \texttt{nil})$$

which in turn reduces to $\bot_{list}(\texttt{rconcat}(\ulcorner t_{i_0}\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil}, \ulcorner t_{i_0+1}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}))$, and then to $\bot_{list}(\ulcorner t_1\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil})$, as wished. □

Lemma 9.3. *Given terms $t_1 \ldots t_n \in \mathcal{T}(\mathcal{F})$, S, and Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_i\urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(\ulcorner t_i\urcorner)$ for all i iff*

$$\begin{aligned}&\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)\\ &\quad .\ \bullet\varphi'_{All(S)}(\varphi_S(\ulcorner t_1\urcorner), \ulcorner t_2\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_1\urcorner :: \texttt{nil}) \longrightarrow\!\!\!\!\!\rightarrow \bot_{list}(\ulcorner t_1\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil})\end{aligned}$$

Proof. Because $\mathbb{T}_{\mathcal{M}}(One(S))$ contains $\mathbb{T}_{\mathcal{M}}(S)$, and by definition of $\mathbb{T}_{\mathcal{M}}(One(S))$, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i\urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(\ulcorner t_i\urcorner)$ for all i iff

$$\begin{aligned}&\varphi'_{One(S)}(\varphi_S(\ulcorner t_i\urcorner), \ulcorner t_{i+1}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_i\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil})\\ &\quad \longrightarrow\!\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)} \varphi'_{One(S)}(\varphi_S(\ulcorner t_{i+1}\urcorner), \ulcorner t_{i+2}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_{i+1}\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil})\end{aligned}$$

for all i. As a result, we deduce

$$\begin{aligned}&\varphi'_{One(S)}(\varphi_S(\ulcorner t_1\urcorner), \ulcorner t_2\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_1\urcorner :: \texttt{nil})\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\varphi'_{One(S)}(\varphi_S(\ulcorner t_n\urcorner), \texttt{nil}, \ulcorner t_n\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil})\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\varphi'_{One(S)}(\bot(\ulcorner t_n\urcorner), \texttt{nil}, \ulcorner t_n\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil})\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\bot_{list}(\texttt{rev}(\ulcorner t_n\urcorner :: \ldots :: \ulcorner t_1\urcorner :: \texttt{nil}))\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)}\ &\bot_{list}(\ulcorner t_1\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil})\end{aligned}$$

as wished. □

Lemma 9.4. *Given terms $t_1 \ldots t_n \in \mathcal{T}(\mathcal{F})$, S, and Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_j\urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(\ulcorner t_j\urcorner)$ for all $j < i_0$ and $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_{i_0}\urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t'_{i_0}\urcorner$ with $\ulcorner t'_{i_0}\urcorner \neq \bot(\ulcorner t_{i_0}\urcorner)$ iff*

$$\begin{aligned}&\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)\\ .\ &\bullet\varphi'_{One(S)}(\varphi_S(\ulcorner t_1\urcorner), \ulcorner t_2\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_1\urcorner :: \texttt{nil}) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t_1\urcorner :: \ldots :: \ulcorner t'_{i_0}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}\end{aligned}$$

Proof. As in the proof of Lemma 9.3, $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t_j\urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(\ulcorner t_j\urcorner)$ for all $j < i_0$ iff

$$\begin{aligned}&\varphi'_{One(S)}(\varphi_S(\ulcorner t_1\urcorner), \ulcorner t_2\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_1\urcorner :: \texttt{nil})\\ &\quad \longrightarrow\!\!\!\!\!\rightarrow_{\mathbb{T}_{\mathcal{M}}(One(S)) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)} \varphi'_{One(S)}(\varphi_S(\ulcorner t_{i_0}\urcorner), \ulcorner t_{i_0+1}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_{i_0}\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil})\end{aligned}$$

This latter term can only reduce to

$$\varphi'_{One(S)}(\ulcorner t'_{i_0}\urcorner, \ulcorner t_{i_0+1}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}, \ulcorner t_{i_0}\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil})$$

which in turn reduces to $\texttt{rconcat}(\ulcorner t_{i_0} - 1\urcorner :: \ldots \ulcorner t_1\urcorner :: \texttt{nil}, \ulcorner t'_{i_0}\urcorner :: \ulcorner t_{i_0+1}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil})$, and then to $\ulcorner t_1\urcorner :: \ldots :: \ulcorner t'_{i_0}\urcorner :: \ldots :: \ulcorner t_n\urcorner :: \texttt{nil}$, as wished. □

Lemma 4.3. *Given a term $t \in \mathcal{T}(\mathcal{F})$, a strategy S and a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$,*

(1) *if $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow u$ with $u \in \mathcal{T}(\mathcal{F})$ or $u = \bot(t)$, then $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner u \urcorner$;*
(2) *if $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow t''$ and $t'' \in \mathcal{T}(\mathcal{F}_{\texttt{appl}})$, then there exists t' such that $\ulcorner t' \urcorner = t''$ and $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow t'$.*
(3) *if $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_S(\ulcorner t \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(t'')$ and $t'' \in \mathcal{T}(\mathcal{F}_{\texttt{appl}})$, then $t'' = \ulcorner t \urcorner$ and $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\!\rightarrow \bot(t)$.*

Proof. The proof is by induction on S. Most cases are straightforward, as the meta-encoding is the same as the encoding in these cases. We only detail the ones with significant changes. In what follows, we write $\mathcal{R}_{\mathrm{M}}$ for $\mathbb{T}_{\mathcal{M}}(S) \cup \mathbb{B}_{\mathcal{M}}(\Gamma)$.

(1) $S = All(S')$.
 (a) $\mathbb{T}(S) \Rightarrow \mathbb{T}_{\mathcal{M}}(S)$.
 Suppose t is a constant c; then $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(c) \longrightarrow\!\!\!\!\!\rightarrow c$. With the meta-encoding, we have

$$\begin{aligned}\varphi_{All(S')}(\texttt{appl}(c,\texttt{nil})) &\longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \texttt{propag}(\texttt{appl}(c,\varphi_S^{list}(\texttt{nil})))\\ &\longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \texttt{propag}(\texttt{appl}(c,\texttt{nil}))\\ &\longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \texttt{appl}(c,\texttt{nil}) = \ulcorner c \urcorner\end{aligned}$$

 The result therefore holds.
 We now suppose that $t = f(t_1,\dots,t_n)$. First, we assume $t' \neq \bot(t)$. By the definition of $\mathbb{T}(S)$, it is possible iff the rule

$$\varphi_{All(S')}(f(x_1,\dots,x_n)) \twoheadrightarrow \varphi_f(\varphi_{S'}(x_1),\dots,\varphi_{S'}(x_n), f(x_1,\dots,x_n))$$

 has been applied, meaning that $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow t'_i$ with $t'_i \neq \bot(t_i)$ for all i. By induction, we have $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t'_i \urcorner$, and $\ulcorner t'_i \urcorner \neq \bot(\ulcorner t_i \urcorner)$. As a result, we have

$$\begin{aligned}&\varphi_{All(S')}(\texttt{appl}(f,\ulcorner t_1 \urcorner :: \dots :: \ulcorner t_n \urcorner :: \texttt{nil}))\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}}\ &\texttt{propag}(\texttt{appl}(f,\varphi_S^{list}(\ulcorner t_1 \urcorner :: \dots :: \ulcorner t_n \urcorner :: \texttt{nil})))\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}}\ &\texttt{propag}(\texttt{appl}(f,\varphi'_S(\varphi_{S'}(\ulcorner t_1 \urcorner),\ulcorner t_2 \urcorner :: \dots :: \ulcorner t_n \urcorner,\ulcorner t_1 \urcorner :: \texttt{nil},\texttt{nil})))\end{aligned}$$

 By Lemma 9.1, we have

$$\begin{aligned}&\texttt{propag}(\texttt{appl}(f,\varphi'_S(\varphi_{S'}(\ulcorner t_1 \urcorner),\ulcorner t_2 \urcorner :: \dots :: \ulcorner t_n \urcorner,\ulcorner t_1 \urcorner :: \texttt{nil},\texttt{nil})))\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}}\ &\texttt{propag}(\texttt{appl}(f,\ulcorner t'_1 \urcorner :: \dots :: \ulcorner t'_n \urcorner))\\ \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}}\ &\texttt{appl}(f,\ulcorner t'_1 \urcorner :: \dots :: \ulcorner t'_n \urcorner) = \ulcorner t' \urcorner\end{aligned}$$

 Assume now $t' = \bot(t)$. By the definition of $\mathbb{T}(S)$, it is possible iff there exists i such that $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_i)$. Let i_0 be the smallest of such i; then for all $j < i_0$, we have $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_j) \longrightarrow\!\!\!\!\!\rightarrow t'_j$ with $t'_j \neq \bot(t_j)$. By induction, we have $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_j \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \ulcorner t'_j \urcorner$ with $\ulcorner t'_j \urcorner \neq \bot(\ulcorner t_j \urcorner)$ for $j < i_0$, and

$\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_{i_0} \urcorner) \longrightarrow\!\!\!\!\rightarrow \bot(\ulcorner t_{i_0} \urcorner)$. Using Lemma 9.2, we have

$$\begin{aligned} & \varphi_{All(S')}(\mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})) \\ \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \ & \mathtt{propag}(\mathtt{appl}(f, \varphi'_S(\varphi_{S'}(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner, \ulcorner t_1 \urcorner :: \mathtt{nil}, \mathtt{nil}))) \\ \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \ & \mathtt{propag}(\mathtt{appl}(f, \bot_{list}(\ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}))) \\ \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \ & \bot(\mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})) = \ulcorner \bot(t) \urcorner \end{aligned}$$

(b) $\mathbb{T}(S) \Rightarrow \mathbb{T}_{\mathcal{M}}(S)$
If t is a constant c, then it is easy to check that $t'' = \ulcorner c \urcorner$. Suppose $t = f(t_1, \ldots, t_n)$. First, we assume $t'' \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$. It is possible only if the rule

$$\varphi'_{All(S')}(x \,@\, \mathtt{appl}(\_,\_), \mathtt{nil}, \_, r\_done) \twoheadrightarrow \mathtt{rev}(x :: r\_done)$$

has been applied. It is possible only if rewriting $\varphi_{S'}(\ulcorner t_i \urcorner)$ produces a term t''_i of the form $\mathtt{appl}(\_,\_)$ for all i, so that $t'' = \mathtt{appl}(\ulcorner f \urcorner, t''_1 :: t''_2 :: \ldots :: t''_n :: \mathtt{nil})$. Only the rules of $\mathbb{T}_{\mathcal{M}}(S')$ may apply to $\varphi_{S'}(\ulcorner t_i \urcorner)$, so we have in fact $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\rightarrow t''_i$ with $t''_i \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$. By induction, there exists t'_i such that $\ulcorner t'_i \urcorner = t''_i$ and $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\rightarrow t'_i$ for all i. From that, we can deduce that $\mathbb{T}(All(S')) \cup \mathbb{B}(\Gamma) \bullet \varphi_{All(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\rightarrow f(t'_1, \ldots, t'_n)$, as wished.
Now assume we obtain $\bot(t'')$ with $t'' \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$. Then the rule

$$\varphi'_{All(S')}(\bot(\_), todo, r\_tried, \_) \twoheadrightarrow \bot_{list}(\mathtt{rconcat}(r\_tried, todo))$$

has been applied, meaning that there exists i such that $\mathbb{T}_{\mathcal{M}}(All(S')) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\rightarrow \bot(t''_i)$ for some t''_i. Only the rules of $\mathbb{T}_{\mathcal{M}}(S')$ may apply to $\varphi_{S'}(\ulcorner t_i \urcorner)$, so we have in fact $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\rightarrow \bot(t''_i)$. By induction, $t''_i = \ulcorner t_i \urcorner$ and $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\rightarrow \bot(t_i)$. From there, we can show that $\mathbb{T}(All(S')) \cup \mathbb{B}(\Gamma) \bullet \varphi_{All(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\rightarrow \bot(f(t_1, \ldots, t_n))$, as wished.

(2) $S = One(S')$.
(a) $\mathbb{T}(S) \Rightarrow \mathbb{T}_{\mathcal{M}}(S)$.
Suppose t is a constant c; then $\mathbb{T}(S) \cup \mathbb{B}(\Gamma) \bullet \varphi_S(c) \longrightarrow\!\!\!\!\rightarrow \bot(c)$. With the meta-encoding, we have

$$\begin{aligned} \varphi_{One(S')}(\mathtt{appl}(c, \mathtt{nil})) & \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \mathtt{propag}(\mathtt{appl}(c, \varphi_S^{list}(\mathtt{nil}))) \\ & \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \mathtt{propag}(\mathtt{appl}(c, \bot_{list}(\mathtt{nil}))) \\ & \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \bot(\mathtt{appl}(c, \mathtt{nil})) = \ulcorner \bot(c) \urcorner \end{aligned}$$

The result therefore holds.
We now suppose that $t = f(t_1, \ldots, t_n)$. First, we assume $t' = \bot(t)$. By the definition of $\mathbb{T}(S)$, it is possible if $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\rightarrow \bot(t_i)$ for all i. By induction, we have $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\rightarrow \ulcorner \bot(t_i) \urcorner$, which in turn implies

$$\begin{aligned} & \varphi_{One(S')}(\mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})) \\ \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \ & \mathtt{propag}(\mathtt{appl}(f, \varphi_S^{list}(\ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}))) \\ \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}_{\mathrm{M}}} \ & \mathtt{propag}(\mathtt{appl}(f, \varphi'_S(\varphi_{S'}(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner, \ulcorner t_1 \urcorner :: \mathtt{nil}))) \end{aligned}$$

By Lemma 9.3, we have

$$\begin{aligned} & \mathtt{propag}(\mathtt{appl}(f, \varphi'_S(\varphi_{S'}(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner, \ulcorner t_1 \urcorner :: \mathtt{nil}))) \\ \twoheadrightarrow_{\mathcal{R}_{\mathrm{M}}} & \mathtt{propag}(\mathtt{appl}(f, \bot_{list}(\ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner))) \\ \twoheadrightarrow_{\mathcal{R}_{\mathrm{M}}} & \bot(\mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner)) = \ulcorner \bot(t) \urcorner \end{aligned}$$

Assume now $t' \neq \bot(t)$. By the definition of $\mathbb{T}(S)$, it is possible if there exists i_0 such that $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow t'_{i_0}$ with $t'_{i_0} \neq \bot(t_{i_0})$, and for all $j < i_0$, we have $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_j) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_j)$. By induction, it means that $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_j \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(\ulcorner t_j \urcorner)$ and $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_{i_0} \urcorner) \longrightarrow\!\!\!\!\!\rightarrow t'_{i_0}$ with $\ulcorner t'_{i_0} \urcorner \neq \bot(\ulcorner t_{i_0} \urcorner)$. Using Lemma 9.4, we have

$$\begin{aligned} & \varphi_{One(S')}(\mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})) \\ \twoheadrightarrow_{\mathcal{R}_{\mathrm{M}}} & \mathtt{propag}(\mathtt{appl}(f, \varphi'_S(\varphi_{S'}(\ulcorner t_1 \urcorner), \ulcorner t_2 \urcorner :: \ldots :: \ulcorner t_n \urcorner, \ulcorner t_1 \urcorner :: \mathtt{nil}))) \\ \twoheadrightarrow_{\mathcal{R}_{\mathrm{M}}} & \mathtt{propag}(\mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t'_{i_0} \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})) \\ \twoheadrightarrow_{\mathcal{R}_{\mathrm{M}}} & \mathtt{appl}(f, \ulcorner t_1 \urcorner :: \ldots :: \ulcorner t'_{i_0} \urcorner :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil}) = \ulcorner t' \urcorner \end{aligned}$$

(b) $\mathbb{T}(S) \Rightarrow \mathbb{T}_{\mathcal{M}}(S)$

If t is a constant c, then it is easy to check that the meta-encoding fails with $\bot(\ulcorner c \urcorner)$. Suppose $t = f(t_1, \ldots, t_n)$. First, we assume we obtain $\bot(t'')$ with $t'' \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$. Then the rule

$$\varphi'_{One(S')}(\bot(\_), \mathtt{nil}, r\_tried) \twoheadrightarrow \bot_{list}(\mathtt{rev}(r\_tried))$$

has been applied, meaning that for all i, we have $\mathbb{T}_{\mathcal{M}}(One(S')) \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(t''_i)$ for some t''_i. Only the rules of $\mathbb{T}_{\mathcal{M}}(S')$ may apply to $\varphi_{S'}(\ulcorner t_i \urcorner)$, so we have in fact $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(t''_i)$ for all i. By induction, $t''_i = \ulcorner t_i \urcorner$ and $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_i)$. From there, we can show that $\mathbb{T}(One(S')) \cup \mathbb{B}(\Gamma) \bullet \varphi_{One(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\rightarrow \bot(f(t_1, \ldots, t_n))$, as wished.

Now we assume $t'' \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$. It is possible only if the rule

$$\varphi'_{One(S')}(x \mathbin{@} \mathtt{appl}(\_, \_), todo, \_ :: r\_tried) \twoheadrightarrow \mathtt{rconcat}(r\_tried, x :: todo)$$

has been applied. It is possible only if there exists i such that $\varphi_{S'}(\ulcorner t_i \urcorner)$ produces a term t''_i of the form $\mathtt{appl}(\_, \_)$, and for all $j < i$, we obtain $\bot(t''_j)$. As a result, we have $t'' = \mathtt{appl}(\ulcorner f \urcorner, \ulcorner t_1 \urcorner :: \ulcorner t_2 \urcorner :: \ldots :: t''_i :: \ldots :: \ulcorner t_n \urcorner :: \mathtt{nil})$. Only the rules of $\mathbb{T}_{\mathcal{M}}(S')$ may apply to $\varphi_{S'}(\ulcorner t_i \urcorner)$, so we have in fact $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_i \urcorner) \longrightarrow\!\!\!\!\!\rightarrow t''_i$ with $t''_i \in \mathcal{T}(\mathcal{F}_{\mathtt{appl}})$ and $\mathbb{T}_{\mathcal{M}}(S') \cup \mathbb{B}_{\mathcal{M}}(\Gamma) \bullet \varphi_{S'}(\ulcorner t_j \urcorner) \longrightarrow\!\!\!\!\!\rightarrow \bot(t''_j)$ for all $j < i$. By induction, there exists t'_i such that $\ulcorner t'_i \urcorner = t''_i$ and $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_i) \longrightarrow\!\!\!\!\!\rightarrow t'_i$, and also $t''_j = \bot(\ulcorner t_j \urcorner)$, $\mathbb{T}(S') \cup \mathbb{B}(\Gamma) \bullet \varphi_{S'}(t_j) \longrightarrow\!\!\!\!\!\rightarrow \bot(t_j)$ for all $j < i$. From that, we can deduce that $\mathbb{T}(One(S')) \cup \mathbb{B}(\Gamma) \bullet \varphi_{One(S')}(f(t_1, \ldots, t_n)) \longrightarrow\!\!\!\!\!\rightarrow f(t_1, \ldots, t'_i, \ldots, t'_n)$, as wished. □

10. Properties of the many-sorted encoding

Lemma 5.6 (Subject reduction)**.** *Consider a many-sorted signature $(\mathcal{S}, \mathcal{F})$, a strategy S, a context Γ such that $\mathcal{FV}ar_{\mathcal{X}}(S) \subseteq \mathcal{D}om_{\mathcal{X}}(\Gamma)$, and the term rewriting system $\mathcal{R} = \mathbb{T}_{\mathcal{S}}(S) \cup \mathbb{B}_{\mathcal{S}}(\Gamma)$ built over the extended signature $(\mathcal{S}, \overline{\mathcal{F}})$. Given a term $t \in \mathcal{T}_s(\overline{\mathcal{F}})$ for some sort $s \in \mathcal{S}$, if $t \longrightarrow_{\mathcal{R}} t'$ then $t' \in \mathcal{T}_s(\overline{\mathcal{F}})$.*

$$
\begin{array}{lrcl}
\textbf{(SE1)} & \mathbb{T}_{\mathcal{S}}(\mathit{Identity}) & = & \bigcup_{s\in\mathcal{S}} \{\ \varphi_{\mathit{Identity}}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle x, \quad \varphi_{\mathit{Identity}}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE2)} & \mathbb{T}_{\mathcal{S}}(\mathit{Fail}) & = & \bigcup_{s\in\mathcal{S}} \{\ \varphi_{\mathit{Fail}}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle \bot(x), \quad \varphi_{\mathit{Fail}}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE3)} & \mathbb{T}_{\mathcal{S}}(l \rightarrowtriangle r) & = & \{\ \varphi_{l\rightarrowtriangle r}(l) \rightarrowtriangle r\ \} \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{l\rightarrowtriangle r}(x \mathbin{@} !^{s} l) \rightarrowtriangle \bot(x), \quad \varphi_{l\rightarrowtriangle r}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE4)} & \mathbb{T}_{\mathcal{S}}(S_1;S_2) & = & \mathbb{T}_{\mathcal{S}}(S_1)\ \cup\ \mathbb{T}_{\mathcal{S}}(S_2) \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{S_1;S_2}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x), \\
 & & & \quad \varphi_{S_1;S_2}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s}), \\
 & & & \quad \varphi_{;}(x \mathbin{@} !^{s}\bot(\_), y^{s}) \rightarrowtriangle x, \quad \varphi_{;}(\bot(y^{s}), x^{s}) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE5)} & \mathbb{T}_{\mathcal{S}}(S_1 \leftarrow\!\!\!+ S_2) & = & \mathbb{T}_{\mathcal{S}}(S_1)\ \cup\ \mathbb{T}_{\mathcal{S}}(S_2) \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{S_1\leftarrow\!\!\!+S_2}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle \varphi_{\leftarrow\!\!\!+}(\varphi_{S_1}(x)), \quad \varphi_{S_1\leftarrow\!\!\!+S_2}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s}), \\
 & & & \quad \varphi_{\leftarrow\!\!\!+}(\bot(x^{s})) \rightarrowtriangle \varphi_{S_2}(x^{s}), \quad \varphi_{\leftarrow\!\!\!+}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle x\ \} \\
\textbf{(SE6)} & \mathbb{T}_{\mathcal{S}}(\mu X\,.\,S) & = & \mathbb{T}_{\mathcal{S}}(S) \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{\mu X.S}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle \varphi_{S}(x), \quad \varphi_{\mu X.S}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s}), \\
 & & & \quad \varphi_{X}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle \varphi_{S}(x), \quad \varphi_{X}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE7)} & \mathbb{T}_{\mathcal{S}}(X) & = & \emptyset \\
\textbf{(SE8)} & \mathbb{T}_{\mathcal{S}}(\mathit{All}(S)) & = & \mathbb{T}_{\mathcal{S}}(S) \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{\mathit{All}(S)}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \bigcup_{c\in\mathcal{F}^{0}} \{\varphi_{\mathit{All}(S)}(c) \rightarrowtriangle c\ \} \\
 & & \bigcup_{f\in\mathcal{F}^{+}} & \{\ \ \varphi_{\mathit{All}(S)}(f(x_1^{s_1},\ldots,x_n^{s_n})) \rightarrowtriangle \\
 & & & \qquad\qquad \varphi_{f}(\varphi_{S}(x_1^{s_1}),\ldots,\varphi_{S}(x_n^{s_n}), f(x_1^{s_1},\ldots,x_n^{s_n})), \\
 & & & \quad \varphi_{f}(x_1 \mathbin{@} !^{s_1}\bot(\_),\ldots,x_n \mathbin{@} !^{s_n}\bot(\_), y^{s}) \rightarrowtriangle f(x_1,\ldots,x_n), \\
 & & & \quad \varphi_{f}(\bot(x_1^{s_1}), x_2^{s_2},\ldots,x_n^{s_n}, x^{s}) \rightarrowtriangle \bot(x^{s}), \\
 & & & \quad \vdots \\
 & & & \quad \varphi_{f}(x_1^{s_1}, x_2^{s_2},\ldots,\bot(x_n^{s_n}), x^{s}) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE9)} & \mathbb{T}_{\mathcal{S}}(\mathit{One}(S)) & = & \mathbb{T}_{\mathcal{S}}(S) \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{\mathit{One}(S)}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \bigcup_{c\in\mathcal{F}^{0}} \{\varphi_{\mathit{One}(S)}(c) \rightarrowtriangle \bot(c)\ \} \\
 & & \bigcup_{f\in\mathcal{F}^{+}} & \{\ \ \varphi_{\mathit{One}(S)}(f(x_1^{s_1},\ldots,x_n^{s_n})) \rightarrowtriangle \varphi_{f_1}(\varphi_{S}(x_1^{s_1}), x_2^{s_2},\ldots,x_n^{s_n})\ \} \\
 & \bigcup_{f\in\mathcal{F}^{+}} & \bigcup_{1\le i\le n} & \{\ \ \varphi_{f_i}(\bot(x_1^{s_1}),\ldots,\bot(x_{i-1}^{s_{i-1}}), x_i \mathbin{@} !^{s_i}\bot(\_), x_{i+1}^{s_{i+1}},\ldots,x_n^{s_n}) \rightarrowtriangle \\
 & & & \qquad\qquad f(x_1^{s_1},\ldots,x_n^{s_n})\ \} \\
 & \bigcup_{f\in\mathcal{F}^{+}} & \bigcup_{1\le i< n} & \{\ \ \varphi_{f_i}(\bot(x_1^{s_1}),\ldots,\bot(x_i^{s_i}), x_{i+1}^{s_{i+1}},\ldots,x_n^{s_n}) \rightarrowtriangle \\
 & & & \qquad\qquad \varphi_{f_{i+1}}(\bot(x_1^{s_1}),\ldots,\bot(x_i^{s_i}), \varphi_{S}(x_{i+1}^{s_{i+1}}), x_{i+2}^{s_{i+2}},\ldots,x_n^{s_n})\ \} \\
 & & \bigcup_{f\in\mathcal{F}^{+}} & \{\ \ \varphi_{f_n}(\bot(x_1^{s_1}),\ldots,\bot(x_n^{s_n})) \rightarrowtriangle \bot(f(x_1^{s_1},\ldots,x_n^{s_n}))\ \} \\
 & & & \\
\textbf{(SE10)} & \mathbb{B}_{\mathcal{S}}(\Gamma; X\colon S) & = & \mathbb{B}_{\mathcal{S}}(\Gamma)\ \cup\ \mathbb{T}_{\mathcal{S}}(S) \\
 & & \bigcup_{s\in\mathcal{S}} & \{\ \ \varphi_{X}(x \mathbin{@} !^{s}\bot(\_)) \rightarrowtriangle \varphi_{S}(x), \quad \varphi_{X}(\bot(x^{s})) \rightarrowtriangle \bot(x^{s})\ \} \\
\textbf{(SE11)} & \mathbb{B}_{\mathcal{S}}(\oslash) & = & \emptyset
\end{array}
$$

Figure 7: Sorted strategy translation. We consider that $f : s_1\times\ldots\times s_n \mapsto s \in \mathcal{F}^{+}_{s}$. The original signature is enriched with $\bot$, $\varphi_{\mathit{Identity}}, \varphi_{\mathit{Fail}}$, $\varphi_{l\rightarrowtriangle r}$, $\varphi_{S_1;S_2}$, $\varphi_{;}$, $\varphi_{S_1\leftarrow\!\!\!+S_2}$, $\varphi_{\leftarrow\!\!\!+}$, $\varphi_{\mu X.S}$, φ_{X}, $\varphi_{\mathit{All}(S)}$, $\varphi_{\mathit{One}(S)} : s \mapsto s$, and $\varphi_{f} : s_1\times\ldots\times s_n\times s \mapsto s$, and $\varphi_{f_{i+1}} : s_1\times\ldots\times s_n \mapsto s$.

Proof. By induction on the structure of the strategy S (and the context Γ) and by cases on the rewrite rule applied in the reduction. We have that $t \longrightarrow_{\mathcal{R}} t'$ iff there exist a rule $l \twoheadrightarrow r \in \mathcal{R}$, $\omega \in \mathcal{P}os(t)$, and a well-sorted substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_{\omega}$; we write $t \longrightarrow_{l \twoheadrightarrow r} t'$ to explicit the applied rule. We first show by cases on the rewrite rule applied in the reduction that the property holds for $\omega = \varepsilon$. It is then easy to conclude by observing that for any term $t\,[u]_{\omega} : s$ with $u : s'$ we also have $t\,[u']_{\omega} : s$ whenever $u' : s'$.

Base case: the applied rewrite rule is one the rules in $\mathbb{T}_{\mathcal{S}}(\mathit{Identity})$, $\mathbb{T}_{\mathcal{S}}(\mathit{Identity})$ or $\mathbb{T}_{\mathcal{S}}(l \twoheadrightarrow r)$.

(1) $t \longrightarrow_{\rho} t'$ with $\rho \in \mathbb{T}_{\mathcal{S}}(\mathit{Identity})$
   (a) ρ corresponds to a rule schema of the form $\varphi_{\mathit{Identity}}(x \,@\, !^{s}\bot(\_)) \twoheadrightarrow x$. In this case $t = \varphi_{\mathit{Identity}}(u)$ with $u : s$ and there exist $v \in \,!^{s}\bot(\_)$ and σ s.t. $\sigma(v) = u$. Then $t' = u$ and thus $t' : s$.
   (b) ρ is a rule of the form $\varphi_{\mathit{Identity}}(\bot(x^{s})) \twoheadrightarrow \bot(x^{s})$. In this case $t = \varphi_{\mathit{Identity}}(\bot(u))$ with $u : s$ and there exists σ s.t. $\sigma(x^{s}) = u$. We have thus $t' = \bot(u)$ and $t' : s$.

(2) $t \longrightarrow_{\rho} t'$ with $\rho \in \mathbb{T}_{\mathcal{S}}(\mathit{Fail})$
   (a) ρ corresponds to a rule schema of the form $\varphi_{\mathit{Fail}}(x \,@\, !^{s}\bot(\_)) \twoheadrightarrow \bot(x)$. In this case $t = \varphi_{\mathit{Fail}}(u)$ with $u : s$ and there exist $v \in \,!^{s}\bot(\_)$ and σ s.t. $\sigma(v) = u$. Then $t' = \bot(u)$ and $t' : s$.
   (b) ρ is a rule of the form $\varphi_{\mathit{Fail}}(\bot(x^{s})) \twoheadrightarrow \bot(x^{s})$. Similar to case 1b.

(3) $t \longrightarrow_{\rho} t'$ with $\rho \in \mathbb{T}_{\mathcal{S}}(l \twoheadrightarrow r)$
   (a) ρ is a rule of the form $\varphi_{l \twoheadrightarrow r}(l) \twoheadrightarrow r$. In this case $t = \varphi_{l \twoheadrightarrow r}(u)$ and $u : s$. Then $l, r : s$ and there exists σ s.t. $\sigma(l) = u$. Then $t' = \sigma(r)$ and $t' : s$.
   (b) ρ corresponds to a rule schema of the form $\varphi_{l \twoheadrightarrow r}(x \,@\, !^{s}l) \twoheadrightarrow \bot(x)$. In this case $t = \varphi_{l \twoheadrightarrow r}(u)$ with $u : s$ and there exist $v \in \,!^{s}l$, $v : s$ and σ s.t. $\sigma(v) = u$. Then $t' = \bot(u)$ and thus $t' : s$.
   (c) ρ is a rule of the form $\varphi_{l \twoheadrightarrow r}(\bot(x^{s})) \twoheadrightarrow \bot(x^{s})$. Similar to case 1b.

Induction case: the applied rewrite rule is one the rules in the encoding of a strategy other than *Identity*, *Fail* or rewrite rule.

(4) $t \longrightarrow_{\rho} t'$ with $\rho \in \mathbb{T}_{\mathcal{S}}(S_1; S_2)$
   (a) ρ corresponds to a rule schema of the form $\varphi_{S_1;S_2}(x \,@\, !^{s}\bot(\_)) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x)$. In this case $t = \varphi_{S_1;S_2}(u)$ with $u : s$ and there exist $v \in \,!^{s}\bot(\_)$ and σ s.t. $\sigma(v) = u$. Then $t' = \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(u)), u)$ and since $\varphi_{S_2}, \varphi_{S_1} : s \mapsto s$ and $\varphi_{;} : s, s \mapsto s$ we have $t' : s$.
   (b) ρ corresponds to a rule schema of the form $\varphi_{;}(x \,@\, !^{s}\bot(\_), y^{s}) \twoheadrightarrow x$. In this case $t = \varphi_{;}(u', u'')$ with $u', u'' : s$. Then $t' = u'$ and thus $t' : s$ using the same reasoning as in case 1a.
   (c) ρ is a rule of the form $\varphi_{S_1;S_2}(\bot(x^{s})) \twoheadrightarrow \bot(x^{s})$. Similar to case 1b.
   (d) ρ is a rule of the form or $\varphi_{;}(\bot(y^{s}), x^{s}) \twoheadrightarrow \bot(x^{s})$. In this case $t = \varphi_{;}(\bot(u'), u'')$ with $u', u'' : s$. Then $t' = \bot(u'')$ and thus $t' : s$ using the same reasoning as in case 1a.
   (e) ρ is one of the rules in $\mathbb{T}_{\mathcal{S}}(S_1) \,\cup\, \mathbb{T}_{\mathcal{S}}(S_2)$. Apply th induction hypothesis.

(5) $t \longrightarrow_{\rho} t'$ with $\rho \in \mathbb{T}_{\mathcal{S}}(S_1 \nleftrightarrow S_2)$. Each of the possible cases is similar to one of the cases for 4.

(6) $t \longrightarrow_{\rho} t'$ with $\rho \in \mathbb{T}_{\mathcal{S}}(\mu X \,.\, S)$. Each of the possible cases is similar to one of the cases for 4.

(7) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}_\mathcal{S}(All(S))$
   (a) ρ is a rule of the form $\varphi_{All(S)}(\bot(x^s)) \twoheadrightarrow \bot(x^s)$. Similar to case 1b.
   (b) ρ is a rule of the form $\varphi_{All(S)}(c) \twoheadrightarrow c$. In this case $t = \varphi_{All(S)}(c)$ with $c : s$ and $t' = c$ and $t' : s$.
   (c) ρ has the form $\varphi_{All(S)}(f(x_1^{s_1}, \ldots, x_n^{s_n})) \twoheadrightarrow \varphi_f(\varphi_S(x_1^{s_1}), \ldots, \varphi_S(x_n^{s_n}), f(x_1^{s_1}, \ldots, x_n^{s_n}))$. In this case $t = \varphi_{All(S)}(f(u_1, \ldots, u_n))$ with $u_1 : s_1, \ldots, u_n : s_n$ if $f : s_1 \times \ldots \times s_n \mapsto s$. Since $\varphi_S : s' \mapsto s'$ for any $s' \in \mathcal{S}$ and $\varphi_f : s_1 \times \ldots \times s_n \times s \mapsto s$ we have $t' : s$.
   (d) ρ corresponds to a rule schema of the form $\varphi_f(x_1 @ !^{s_1}\bot(\_), \ldots, x_n @ !^{s_n}\bot(\_), y^s) \twoheadrightarrow f(x_1, \ldots, x_n)$. In this case $t = \varphi_f(u_1, \ldots, u_n, u)$ with $u_1 : s_1, \ldots, u_n : s_n, u : s$ if $f : s_1 \times \ldots \times s_n \mapsto s$ and for all $i = 1 \ldots n$ there exist $v_i \in !^{s_i}\bot(\_)$ and σ s.t. $\sigma(v_i) = u_i$. Then $t' = f(u_1, \ldots, u_n)$ and thus $t' : s$.
   (e) ρ is a rule of the form or $\varphi_f(x_1^{s_1}, \ldots, \bot(x_i^{s_i}), \ldots, x_n^{s_n}, x^s) \twoheadrightarrow \bot(x^s)$. In this case $t = \varphi_f(u_1, \ldots, \bot(u_i), \ldots, u_n, u)$ with $u_1 : s_1, \ldots, u_n : s_n, u : s$ if $f : s_1 \times \ldots \times s_n \mapsto s$. Then $t' = \bot(u)$ and thus $t' : s$.
   (f) ρ is one of the rules in $\mathbb{T}_\mathcal{S}(S)$. Apply th induction hypothesis.

(8) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}_\mathcal{S}(One(S))$. Each of the possible cases is similar to one of the cases for 7.

(9) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{B}_\mathcal{S}(\Gamma; X : S)$. Each of the possible cases is similar to one of the cases for 6, for example. □

Lemma 5.7 (Equivalent reductions)**.** *Consider a many-sorted signature $(\mathcal{S}, \mathcal{F})$, a strategy S, a context Γ such that $\mathcal{FV}ar_\mathcal{X}(S) \subseteq \mathcal{D}om_\mathcal{X}(\Gamma)$, and the term rewriting systems $\mathcal{R} = \mathbb{T}(S) \cup \mathbb{B}(\Gamma)$ and $\mathcal{R}_\mathcal{S} = \mathbb{T}_\mathcal{S}(S) \cup \mathbb{B}_\mathcal{S}(\Gamma)$ built over the extended signature $(\mathcal{S}, \overline{\mathcal{F}})$. Given a term $t \in \mathcal{T}_\mathcal{S}(\overline{\mathcal{F}})$, $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$ iff $t \longrightarrow_\mathcal{R} t'$.*

Proof. Every rewrite rule in $\mathcal{R}_\mathcal{S}$ is also included in $\mathcal{R}$ and thus, if $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$ then $t \longrightarrow_\mathcal{R} t'$.

For the other direction we proceed by induction on the structure of the strategy S (and the context Γ) and by cases on the rewrite rule applied in the reduction. For simplicity, we consider reduction at the top position but as shown in the proof of Lemma 5.6 this generalises immediately for any application position. We consider in what follows that $t \in \mathcal{T}_s(\overline{\mathcal{F}})$ for some $s \in \mathcal{S}$.

Base case: the applied rewrite rule is one the rules in $\mathbb{T}(Identity)$, $\mathbb{T}(Identity)$ or $\mathbb{T}(l \twoheadrightarrow r)$.

(1) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}(Identity)$
   (a) ρ is one of the rules corresponding to the rule schema $\varphi_{Identity}(x @ !\bot(\_)) \twoheadrightarrow x$ and thus of the form $\varphi_{Identity}(f(x_1, \ldots, x_n)) \twoheadrightarrow f(x_1, \ldots, x_n)$ with $f \in \mathcal{F}$. In this case $t = \varphi_{Identity}(f(t_1, \ldots, t_n))$ and thus $f \in \mathcal{F}_s$. This rule also corresponds to the rule schema $\varphi_{Identity}(x @ !^s\bot(\_)) \twoheadrightarrow x$ from the sorted translation and thus $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$.
   (b) ρ is the rule $\varphi_{Identity}(\bot(x)) \twoheadrightarrow \bot(x)$. The rule $\varphi_{Identity}(\bot(x^s)) \twoheadrightarrow \bot(x^s)$ from the sorted translation applies also to t which has necessarily the form $\varphi_{Identity}(\bot(u))$ with $u : s$ and thus $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$.

(2) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}(Fail)$. We proceed similarly as for the case 1.

(3) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}(l \twoheadrightarrow r)$
   (a) ρ is one of the rules corresponding to the rule schema $\varphi_{l \twoheadrightarrow r}(x @ !l) \twoheadrightarrow \bot(x)$ and thus of the form $\varphi_{l \twoheadrightarrow r}(f(u_1, \ldots, u_n)) \twoheadrightarrow \bot(f(u_1, \ldots, u_n))$ with $f \in \mathcal{F}$. In this case $t = \varphi_{l \twoheadrightarrow r}(f(t_1, \ldots, t_n))$ and thus $f \in \mathcal{F}_s$. Consequently, the rule ρ corresponds

also to the rule schema $\varphi_{l \twoheadrightarrow r}(x \,@\, !^s l) \twoheadrightarrow \bot(x)$ from the sorted translation and thus $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$.
(b) ρ is the rule $\varphi_{l \twoheadrightarrow r}(l) \twoheadrightarrow r$. Since the rule is also present in the sorted translation then $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$.
(c) ρ is the rule $\varphi_{l \twoheadrightarrow r}(\bot(x)) \twoheadrightarrow \bot(x)$. Similar to the case 1b.

Induction case: the applied rewrite rule is one the rules in the encoding of a strategy other than *Identity*, *Fail* or rewrite rule. When the applied rule is not in $\mathbb{T}(All(S))$ or $\mathbb{T}(One(S))$ we can conclude either by induction or similarly to one of the above cases.

(4) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}(All(S))$. Once again we can proceed either by induction or similarly to one of the above cases except for one case:
   (a) ρ corresponds to a rule schema $\varphi_f(x_1 \,@\, !\bot(\_), \ldots, x_n \,@\, !\bot(\_), \_) \twoheadrightarrow f(x_1, \ldots, x_n)$ and thus is a rewrite rule of the form $\varphi_f(f_1(x_1^1, \ldots, x_m^1), \ldots, f_n(x_1^n, \ldots, x_p^n), x) \twoheadrightarrow f(f_1(x_1^1, \ldots, x_m^1), \ldots, f_n(x_1^n, \ldots, x_p^n))$ with $f_1, \ldots, f_n, f \in \mathcal{F}$. Since the rewrite rule can be applied to the well-sorted term t then $f_1 \in \mathcal{F}_{s_1}, \ldots, f_n \in \mathcal{F}_{s_n}$, $f \in \mathcal{F}_s$ and consequently, this rewrite rule also corresponds to the rule schema $\varphi_f(x_1 \,@\, !^{s_1}\bot(\_), \ldots, x_n \,@\, !^{s_n}\bot(\_), y^s) \twoheadrightarrow f(x_1, \ldots, x_n)$ from the sorted translation. Thus $t \longrightarrow_{\mathcal{R}_\mathcal{S}} t'$.

(5) $t \longrightarrow_\rho t'$ with $\rho \in \mathbb{T}_\mathcal{S}(One(S))$. Similar to the cases for 4. □

Theorem 5.8 (Simulation)**.** *Given a a many-sorted signature $(\mathcal{S}, \mathcal{F})$, a strategy S, two terms $t, t' \in \mathcal{T}_\mathcal{S}(\mathcal{F})$, and a context Γ such that $\mathcal{FV}ar_\mathcal{X}(S) \subseteq \mathcal{D}om_\mathcal{X}(\Gamma)$,*
(1) $\Gamma \vdash S \circ t \Longrightarrow t'$ *iff* $\mathbb{T}_\mathcal{S}(S) \cup \mathbb{B}_\mathcal{S}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\rightarrow t'$,
(2) $\Gamma \vdash S \circ t \Longrightarrow$ `Fail` *iff* $\mathbb{T}_\mathcal{S}(S) \cup \mathbb{B}_\mathcal{S}(\Gamma) \bullet \varphi_S(t) \longrightarrow\!\!\!\!\rightarrow \bot(t)$

Proof. Follows immediately from Lemma 5.7 and Theorem 3.5. □