Correctness of Program Transformations as a Termination Problem*

Conrad Rau, David Sabel, and Manfred Schmidt-Schauß

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

Abstract. The diagram-based method to prove correctness of program transformations includes the computation of (critical) overlappings between the analyzed program transformation and the (standard) reduction rules which result in so-called forking diagrams. Such diagrams can be seen as rewrite rules on reduction sequences which abstract away the expressions and allow additional expressive power, like transitive closures of reductions. In this paper we clarify the meaning of forking diagrams using interpretations as infinite term rewriting systems. We then show that the termination problem of forking diagrams as rewrite rules can be encoded into the termination problem for conditional integer term rewriting systems, which can be solved by automated termination provers. Since the forking diagrams can be computed automatically, the results of this paper are a big step towards a fully automatic prover for the correctness of program transformations.

1 Introduction

This work is motivated from proving correctness of program transformations in program calculi that model core languages of functional programming languages. For instance, Haskell [13] is modeled by the calculus *LR* [21], Concurrent Haskell [14] is modeled by the calculus CHF [19], and Alice ML[1] is modeled by the calculus $\lambda(\mathsf{fut})$ [12,11]. A *program transformation* transforms one program into another one. It is correct if the semantics of the program is unchanged, i.e. the programs before and after the transformation are semantically equivalent. *Correctness* of program transformations plays an important role in several fields of computer science: Optimizations applied while compiling programs are program transformations and their correctness thus ensures correct compilation. For software verification programs are transformed or simplified to show properties of programs, of course these transformations must be correct. In code refactoring programs are redesigned, but the semantics of the programs must not be changed, i.e. the transformations must be correct.

As semantics (or equality) of programs we choose *contextual equivalence* [10,15], since it is a natural notion of program equivalence which can directly

* This work was supported by the DFG under grant SCHM 986/9-1.

[1] http://www.ps.uni-saarland.de/alice/

be defined on top of the operational semantics. Two programs are contextually equivalent if their termination behavior is indistinguishable if they are used as subprograms in any surrounding larger program (which are called the *contexts*, denoted by C). For deterministic and expressive programming languages it is sufficient to observe whether the program's execution terminates successfully, since there are enough contexts to discriminate obviously different programs.

Proving two expressions to be contextually equivalent starting from the definition is inconvenient, since all program contexts must be considered. Several methods and theoretical tools have been developed to ease the proofs, however, depending on properties of the program calculus. In this paper we concentrate on the so-called *diagram-based method* to prove correctness of program transformations, which was successfully used for several calculi, e.g., [7,9,11,21,18,19]. Diagram uses that are similar to ours also appear in [1]. Related work on diagram methods is [22], who aim at meaning preservation and make a distinction between standard reduction and transformation. Also [8] propose the use of diagrams to prove meaning preservation during compilation.

The diagram method, as we use it, is syntactic in nature, where the steps can roughly be described as follows: Let $\overset{T}{\Rightarrow}$ be a program transformation, i.e. a binary relation on expressions. First a set of overlappings between the standard reduction of the calculus and the transformation $\overset{T}{\Rightarrow}$ is computed, resulting in a so-called (finite) *complete set of forking diagrams* for $\overset{T}{\Rightarrow}$. The second task is to show that for all expressions e_1, e_2, and contexts C such that $C[e_1] \overset{T}{\Rightarrow} C[e_2]$: The program $C[e_1]$ converges, if and only if, the program $C[e_2]$ converges. Starting with a successful reduction sequence (evaluation) for $C[e_1]$ (or $C[e_2]$, resp.), we construct a successful reduction sequence for $C[e_2]$ ($C[e_1]$, resp.) by an induction, where the forking diagrams are used like a (non-deterministic) rewriting system and the normal form is the desired evaluation.

Our current research goal is to automate the manual proofs in the diagram method. We already proposed an extended unification algorithms which performs the computation of the forking diagrams for the call-by-need lambda calculus and for the above mentioned calculus LR [16,17]. We will show that the missing part of the correctness proof, i.e. using the diagrams and induction, can be performed by showing (innermost) termination of a term rewriting system that can be constructed from the forking diagrams. The termination proof can then be automated using termination provers like AProVE [5], TTT2 [6], and CiME [2].

In this paper we rigorously analyze the use of forking diagrams as rewriting problems on reduction sequences. The goal is twofold: to encode the induction proofs as a termination proof of TRSs and also to clarify the intermediate steps thereby showing in a general way that the encoding method is sound. The forking diagrams are denoted by an expressive language, also permitting transitive closure. They only speak about the arrows (perhaps labeled) of a reduction, and completely abstract away the expressions. To show that the encoding is correct, we provide a link to the concrete reductions on expressions, which requires two levels of abstractions. Finally, we will show that the termination problem can be expressed (or encoded) by extended term rewriting systems, which are con-

ditional integer term rewrite systems (ITRS) (see e.g., [4]). Since AProvE can only show innermost termination of ITRS, our encodings are carefully designed to require innermost termination only. We applied these encodings to the diagrams of the calculus LR and the manually computed forking diagrams in [21] and used AProVE to show termination (and thus correctness) of several program transformations automatically.

Structure of the Paper. In Sect. 2 we introduce the notions of a program calculus, contextual equivalence, and correct program transformations. In Sect. 3 we explain the diagram-based method, and introduce several abstractions for those diagrams and the corresponding rewriting systems. Our main result is obtained in Theorem 3.27 showing that correctness of program transformations can be encoded as a termination problem. In Sect. 4 we apply our techniques to transformations of the calculus LR and show the step-wise encoding of the diagrams of two transformations into an integer term rewriting system for which AProVE can automatically prove termination. Finally, we conclude in Sect. 5.

2 Calculi and Program Transformations

In this section we introduce the notion of a program calculus, contextual equivalence, and correctness of program transformations.

Definition 2.1. *A* program calculus *is a tuple* $(\mathcal{E}, \mathcal{C}, \overset{sr}{\Rightarrow}, \mathcal{A}, \mathcal{L})$ *where* $\mathcal{E}$ *is the set of* expressions*,* $\mathcal{C}$ *is the set of* contexts*, where* $C \in \mathcal{C}$ *is a function from* $\mathcal{E}$ *into* $\mathcal{E}$*,* $\overset{sr}{\Rightarrow} \subseteq \mathcal{E} \times \mathcal{E} \times \mathcal{L}$ *is a reduction relation,* $\mathcal{A} \subseteq \mathcal{E}$ *is a set of* answers*, and* $\mathcal{L}$ *is a finite set of labels. We assume that there is a context* $[\cdot] \in \mathcal{C}$*, such that* $[e] = e$ *for all* $e \in \mathcal{C}$*, and that* $\mathcal{C}$ *is a monoid with* $[\cdot]$ *as unit, such that* $(C_1C_2)[e] = C_1[C_2[e]]$*. We write* $\overset{sr,l}{\Longrightarrow} \subseteq \overset{sr}{\Rightarrow}$ *for reductions with label* $l \in \mathcal{L}$*.*

The contexts $\mathcal{C}$ consist of all expressions of $\mathcal{E}$ where one subexpression is replaced by the context hole. The reduction $\overset{sr}{\Rightarrow}$ is a small step reduction as the **s**tandard **r**eduction of the calculus, where the labels distinguish different kinds of reductions. We do not require that answers $a \in \mathcal{A}$ are $\overset{sr}{\Rightarrow}$-irreducible. The converse relation is always written by reversing the arrows.

Example 2.2. The program calculus $LR = (\mathcal{E}, \mathcal{C}, \overset{sr}{\Rightarrow}, \mathcal{A}, \mathcal{L})$ [21] is an extended call-by-need lambda calculus where expressions $\mathcal{E}$ comprise abstractions, applications, data-constructors, case-expressions, `letrec` for recursive shared bindings, and `seq` for strict evaluation. $\mathcal{C}$ is the set of contexts. The standard reduction $\overset{sr}{\Rightarrow}$ of LR is called *normal order reduction* denoted by $\overset{n}{\Rightarrow}$ and the answers are so-called *weak head normal forms*. The set of labels $\mathcal{L}$ are the names of the standard reductions, e.g. *seq*, *lbeta*, and *llet*.

The *evaluation* of a program expression $e \in \mathcal{E}$ is a sequence of standard reduction steps to some answer $a \in \mathcal{A}$, i.e. $e \overset{sr,*}{\Longrightarrow} a$, where $\overset{sr,*}{\Longrightarrow}$ denotes the reflexive-transitive closure of $\overset{sr}{\Rightarrow}$. If such an evaluation exists, then we write $e{\Downarrow}$ and say e *converges*, otherwise we write $e{\Uparrow}$ and say e *diverges*. The semantics of expressions is given by contextual equivalence:

Definition 2.3. *For a program calculus* $(\mathcal{E}, \mathcal{C}, \overset{sr}{\Rightarrow}, \mathcal{A}, \mathcal{L})$ contextual preorder $\leq_c$ *and* contextual equivalence $\sim_c$ *on expressions* $e, e' \in \mathcal{E}$ *are defined as follows:*

$$e \leq_c e' \iff \forall C \in \mathcal{C} : C[e]\Downarrow \implies C[e']\Downarrow \quad \textit{and} \quad e \sim_c e' \iff e \leq_c e' \wedge e' \leq_c e.$$

Definition 2.4. *A* program transformation $\overset{T}{\Rightarrow} \subseteq (\mathcal{E} \times \mathcal{E})$ *is a binary relation on expressions, and* $\overset{T}{\Rightarrow}$ *is called* correct *iff* $\overset{T}{\Rightarrow} \subseteq \sim_c$

Definition 2.5. *A relation* $R \subseteq \mathcal{E} \times \mathcal{E}$ *is called* convergence-preserving *iff* $(e, e') \in R \wedge e\Downarrow \implies e'\Downarrow$. *If* R *and its inverse relation* R^- *are convergence-preserving then we say* R *is* convergence-equivalent.

Often, the correctness proof for $\overset{T}{\Rightarrow}$ is done by applying the diagram method to a modified transformation $\overset{T'}{\Rightarrow}$, and then using theorems of the calculus.

Definition 2.6. *A program transformation* $\overset{T'}{\Rightarrow}$ *is* CP-sufficient *for a program transformation* $\overset{T}{\Rightarrow}$ *iff convergence preservation of* $\overset{T'}{\Rightarrow}$ *implies* $\overset{T}{\Rightarrow} \subseteq \leq_c$.

For a transformation $\overset{T}{\Rightarrow}$, let $\overset{\mathcal{C}(T)}{\Longrightarrow} := \{(C[e], C[e']) \mid e \overset{T}{\Rightarrow} e', C \in \mathcal{C}\}$. Then the transformation $\overset{\mathcal{C}(T)}{\Longrightarrow}$ is CP-sufficient for $\overset{T}{\Rightarrow}$. However, this still requires to inspect all contexts $C \in \mathcal{C}$ for proving correctness of transformation $\overset{T}{\Rightarrow}$. In many calculi a so-called context lemma holds (see e.g. [3,20]) which shows that the relation $\overset{\mathcal{R}(T)}{\Longrightarrow} := \{(R[e], R[e']) \mid e \overset{T}{\Rightarrow} e', R \in \mathcal{R}\}$ is CP-sufficient for $\overset{T}{\Rightarrow}$, where $\mathcal{R} \subset \mathcal{C}$ are so-called reduction contexts. The following corollary describes the method for proving correctness. It follows directly from Definitions 2.3 and 2.6.

Corollary 2.7. *If* $\overset{T'}{\Rightarrow}$ *is CP-sufficient for* $\overset{T}{\Rightarrow}$, $\overset{T''}{\Longleftarrow}$ *is CP-sufficient for* $\overset{T}{\Longleftarrow}$, *and* $\overset{T'}{\Rightarrow}$ *and* $\overset{T''}{\Longleftarrow}$ *are both convergence-preserving, then* $\overset{T}{\Rightarrow}$ *is correct.*

Proving convergence preservation of a transformation $\overset{T}{\Rightarrow}$ requires as a base case to inspect what happens if an answer $a \in \mathcal{A}$ is transformed by $\overset{T}{\Rightarrow}$.

Definition 2.8. *A program transformation* $\overset{T}{\Rightarrow}$ *is called* answer-preserving (weakly answer-preserving), *if* $a \in \mathcal{A}$ *and* $a \overset{T}{\Rightarrow} e$ *imply* $e \in \mathcal{A}$ ($e\Downarrow$, *respectively).*

3 Proving Correctness of Program Transformations

Throughout this section we assume that a program calculus $(\mathcal{E}, \mathcal{C}, \overset{sr}{\Rightarrow}, \mathcal{A}, \mathcal{L})$ is given. In this section we explain our diagram-based method to prove convergence preservation of a transformation $\overset{T}{\Rightarrow}$. For showing that $e_0\Downarrow$ is implied by $e_1 \overset{T}{\Rightarrow} e_0$ and $e_1\Downarrow$, we start with a sequence of reductions $a \overset{sr,l_n}{\Longleftarrow} e_n \overset{sr,l_{n-1}}{\Longleftarrow} \ldots \overset{sr,l_1}{\Longleftarrow} e_1 \overset{T}{\Rightarrow} e_0$ where $a \in \mathcal{A}$ and *rewrite* this sequence resulting in a sequence $a' \overset{sr,l'_m}{\Longleftarrow} e'_m \overset{sr,l'_{m-1}}{\Longleftarrow} \ldots \overset{sr,l'_0}{\Longleftarrow} e_0$ (with $a' \in \mathcal{A}$) validating that $e_0\Downarrow$ holds. If this is possible for all $e_1 \overset{T}{\Rightarrow} e_0$ then convergence preservation of $\overset{T}{\Rightarrow}$ is proven.

Definition 3.1. *Let* $\{\overset{T_1}{\Longrightarrow}, \ldots, \overset{T_k}{\Longrightarrow}\}$ *be a set of program transformations. Then a* concrete reduction sequence (RS) *is a string of elements in* $\overset{sr}{\Longleftarrow} \cup \bigcup_{1 \leq i \leq k}(\overset{T_i}{\Longrightarrow} \times \{i\}) \cup \{(a, a, \underline{id}) \mid a \in \mathcal{A}\}$ *with the restrictions that* $(a, a, \underline{id})$ *can only be the leftmost element, and that two subsequent elements* $(e_1, e_2, d)(e_3, e_4, d')$ *are only permitted if* $e_2 = e_3$*. We write* $e_1 \overset{sr,l}{\Longleftarrow} e_2$ *for* (e_1, e_2, l)*,* $e_1 \overset{T_i}{\Longrightarrow} e_2$ *for* (e_1, e_2, i)*, and* $a \underline{id} a$ *for* $(a, a, \underline{id})$*. An RS is a* converging concrete reduction sequence (cRS) *if its leftmost reduction is of the form* $a \underline{id} a$*. Let* cRS *be the set of all cRSs.*

We write RSs like reduction sequences, e.g. $e_1 \overset{sr,l}{\Longleftarrow} e_2 \; e_2 \overset{T}{\Rightarrow} e_3$ is written as $e_1 \overset{sr,l}{\Longleftarrow} e_2 \overset{T}{\Rightarrow} e_3$. A *rewrite rule* on RSs is a rule $S_1 \rightsquigarrow S_2$ where S_1, S_2 are RSs.

Definition 3.2. *Let* D *be a set of rewrite rules on RSs. Then the pair* $(\mathsf{cRS}, \overset{D}{\longrightarrow})$ *is a string rewrite system, called a* concrete rewrite system on RSs *(CRSRS).*

3.1 Abstract Reduction Sequences

For reasoning we use abstract reduction sequences (ARS), which abstract away concrete expressions, and where abstract symbols represent the reductions and transformations, and a special constant A represents answers. To distinguish concrete and abstract reductions we use solid lines on the abstract level (i.e. $\overset{sr}{\longrightarrow}$ instead of $\overset{sr}{\Longrightarrow}$), in contrast to doubly lined-arrows on the concrete level.

We also provide an interpretation of ARSs which maps them into concrete sequences. Note that there may be ARSs without a corresponding RS. We define two variants of abstract reduction sequences, those that must start with an answer and a more general variant which may start with any expression.

Definition 3.3. *An* abstract reduction sequence *(ARS) is a finite sequence* $I_n \ldots I_1$*, and a* converging ARS *(cARS) is a finite sequence* $AI_n \ldots I_1$ *where* A *is a constant representing any answer,* $n \geq 0$*. The symbol* I_j *may either be the symbol* $\overset{sr,l}{\longleftarrow}$ *with* $l \in \mathcal{L}$ *representing a (labeled) standard reduction, the symbol* $\overset{sr,x}{\longleftarrow}$ *where* x *is a variable,* $\overset{sr,\tau}{\longleftarrow}$ *with* $\tau \notin \mathcal{L}$ *where* τ *represents a union of labels, or the symbol* $\overset{T_i}{\longrightarrow}$ *representing transformation* T_i*. Any symbol can also be extended by* $+$ *representing the transitive closure of the corresponding reduction or transformation. Symbols that have sr as a part are called* sr-symbols*, and other symbols are called* transformation-symbols*.*

An ARS or cARS that does not contain variables is called ground*, and a ground ARS or cARS is called* simple *if there is no occurrence of* $+$*. An ARS or cARS that does not contain* $\overset{sr,\tau}{\longleftarrow}$*-symbols is called* τ*-free.*

Definition 3.4. *Let* S *be a simple ARS (or* S *be a simple cARS, resp.) and* $M \subseteq \mathcal{L}$ *be a set of labels. The* interpretation w.r.t. M *is the set* $\mathcal{I}_M(S)$ *of RSs (cRSs, resp.) defined recursively by the following cases, where* S_1, S_2 *are non-empty sequences,* ϵ *denotes the empty sequence, and* $e_1 \bowtie e_2$ *means a RS that starts with expression* e_1 *and ends with expression* e_2*.*

$$\mathcal{I}_M(\epsilon) := \emptyset \qquad \mathcal{I}_M(A) := \{a \xrightarrow{id} a \mid a \in \mathcal{A}\}$$
$$\mathcal{I}_M(\xleftarrow{sr,l}) := \{e_1 \xleftarrow{sr,l} e_2 \mid e_2 \xRightarrow{sr,l} e_1\} \qquad \mathcal{I}_M(\xrightarrow{T_i}) := \{e_1 \xRightarrow{T_i} e_2 \mid e_1 \xRightarrow{T_i} e_2\}$$
$$\mathcal{I}_M(\xleftarrow{sr,\tau}) := \{e_1 \xleftarrow{sr,l} e_2 \mid e_2 \xRightarrow{sr,l} e_1, l \in M\}$$
$$\mathcal{I}_M(S_1S_2) := \{e_1 \bowtie e_2 \bowtie' e_3 \mid e_1 \bowtie e_2 \in \mathcal{I}_M(S_1), e_2 \bowtie' e_3 \in \mathcal{I}_M(S_2)\}$$

3.2 Rewriting by Forking and Answer Diagrams

ARSs are used in the so-called forking diagrams [7,21], which represent the overlappings of transformation steps with standard reductions on the abstract level.

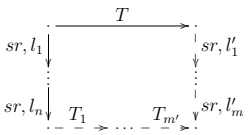
General forking diagrams are a *finite* representation of all overlappings between a transformation step and a standard reduction step, and are suitable for automated encoding. They may contain + for transitive closure and label-variables. For clarifying their meaning we introduce simple forking diagrams (without label-variables and transitive closures, but with τ).

Definition 3.5. *A* general forking diagram *for a transformation $\xRightarrow{T}$ is a rewrite rule $S_L \rightsquigarrow S_R$ where S_L, S_R are τ-free ARSs and:*

- *S_L is of the form $I_n \dots I_1 \xrightarrow{T}$ with $n \geq 0$ and all I_i are sr-symbols.*
- *S_R is of the form $J_m \dots J_1 I'_{m'} \dots I'_1$ with $m, m' \geq 0$ where all J_i are transformation-symbols and I'_i are sr-symbols.*

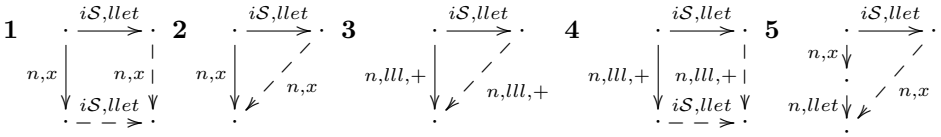
A simple forking diagram *$S_L \rightsquigarrow S_R$ is defined like a forking diagram where S_L and S_R are simple ARSs (which are not necessarily τ-free).*

$\xleftarrow{sr,l_n} \dots \xleftarrow{sr,l_1} \xrightarrow{T} \rightsquigarrow \xrightarrow{T_1} \dots \xrightarrow{T_{m'}} \xleftarrow{sr,l'_m} \dots \xleftarrow{sr,l'_1}$ is a forking diagram as shown to the right, where the left hand side of the rule (the solid arrows) form a fork and the right hand side (the dashed arrows) join the fork.



Example 3.6. The transformation $\xRightarrow{llet}$ of the calculus *LR* (Example 2.2) is defined by two rules, where *Env* is an environment of the form $y_1 = e_1, \dots, y_n = e_n$:
$\texttt{letrec}\ Env_1\ \texttt{in}\,(\texttt{letrec}\ Env_2\ \texttt{in}\ e) \xRightarrow{llet} \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$
$\texttt{letrec}\ Env_1, y = (\texttt{letrec}\ Env_2\ \texttt{in}\ e')\,\texttt{in}\,e \xRightarrow{llet} \texttt{letrec}\ Env_1, Env_2, y = e'\,\texttt{in}\,e$

A (complete) set of general forking diagrams for the transformation $\xRightarrow{i\mathcal{S},llet}$ (which is CP-sufficient for the transformation $\xRightarrow{llet}$) consists of five diagrams:



Definition 3.7. *Let D be a set of rewrite rules of the form $S_L \rightsquigarrow S_R$ where S_L, S_R are simple ARSs. Let $\mathsf{cARS}(D)$ be the set of simple cARSs that can be built by the symbols occurring in D. Then the string rewriting system $(\mathsf{cARS}(D), \xrightarrow{D})$ is called a* simple rewrite system on abstract reduction sequences *(SRSARS).*

On the concrete level, the interpretation of a simple forking diagram is a set of rewrite rules on (concrete) reduction sequences:

Definition 3.8. *The interpretation* $\mathcal{I}_M(S_L \rightsquigarrow S_R)$ *of a simple forking diagram* $S_L \rightsquigarrow S_R$ *w.r.t. a set of labels* $M \subseteq \mathcal{L}$ *is defined as*

$$\mathcal{I}_M(S_L \rightsquigarrow S_R) := \{e_1 \bowtie e_2 \rightsquigarrow e_1 \bowtie' e_2 \mid e_1 \bowtie e_2 \in \mathcal{I}_M(S_L), e_1 \bowtie' e_2 \in \mathcal{I}_M(S_R)\}$$

We will also interpret general forking diagrams as sets of simple forking diagrams (and thus also as rewrite rules on RSs using $\mathcal{I}_M$). We first introduce the notion of a variable interpretation which assigns concrete labels or τ to symbols $\xleftarrow{sr,x}$ and the notion of an expansion which unfolds the symbols containing a $+$ for the transitive closure of a reduction or transformation.

Definition 3.9. *A* simple expansion Exp_k *(where* $k \geq 1$*) expands symbols as follows:* $Exp_k(\xrightarrow{T_i,+}) = \underbrace{\xrightarrow{T_i} \dots \xrightarrow{T_i}}_{k \text{ times}}$ *and* $Exp_k(\xleftarrow{sr,l,+}) = \underbrace{\xleftarrow{sr,l} \dots \xleftarrow{sr,l}}_{k \text{ times}}$ *where* $l \in \mathcal{L} \cup \{\tau\}$*, and* $Exp_k(I) = I$ *otherwise. For a ARS (or cARS)* $S = I_n \dots I_1$ *we define* $Exp_\pi(S) := Exp_{\pi(1)}(I_n) \dots Exp_{\pi(n)}(I_1)$ *where* $\pi : \mathbb{N} \to \mathbb{N}$*, and* Exp_π *denotes the* expansion *for* π*. For a set of labels* $M \subseteq \mathcal{L}$ *a* variable interpretation $V_{\neg M}$ *maps any* $\xleftarrow{sr,x}$*-symbol to a symbol* $\xleftarrow{sr,l}$ *where* $l = V_{\neg M}(x) \in (\mathcal{L} \setminus M) \cup \{\tau\}$*.*

General forking diagrams are interpreted as a set of simple forking diagrams:

Definition 3.10. *For a general forking diagram* $S_L \rightsquigarrow S_R$ *and* $M \subseteq \mathcal{L}$ *the translation* $\mathcal{J}_M(S_L \rightsquigarrow S_R)$ *is a set of simple forking diagrams* $\mathcal{J}_M(S_L \rightsquigarrow S_R) :=$

$$\left\{ Exp_\pi(V_{\neg M}(S_L)) \rightsquigarrow Exp_{\pi'}(V_{\neg M}(S_R)) \;\middle|\; \begin{array}{l} V_{\neg M} \text{ is a variable interpretation for} \\ M,\ Exp_\pi \text{ and } Exp_{\pi'} \text{ are expansions} \end{array} \right\}$$

We also use $\mathcal{J}_M$ *for sets of forking diagrams, where the resulting sets are joined.*

Example 3.11. Let D be the third diagram from Example 3.6. For $\mathcal{L} = \{lll, llet, seq, \dots\}$ and $M = \mathcal{L} \setminus \{lll, llet\}$ the translation $\mathcal{J}_M(D)$ is $\{\xleftarrow{n,lll} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xleftarrow{n,lll}, \xleftarrow{n,lll} \xleftarrow{n,lll} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xleftarrow{n,lll}, \xleftarrow{n,lll} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xleftarrow{n,lll} \xleftarrow{n,lll}, \dots\}$. If D is the second diagram from Example 3.6 then $\mathcal{J}_M(D) = \{\xleftarrow{n,lll} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xleftarrow{n,lll}, \xleftarrow{n,llet} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xleftarrow{n,llet}, \xleftarrow{n,\tau} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xleftarrow{n,\tau}\}$.

With $DF(\overset{T}{\Rightarrow})$ we denote a set of forking diagrams for a transformation $\overset{T}{\Rightarrow}$.

Definition 3.12. *A set of forking diagrams* $DF(\overset{T}{\Rightarrow})$ *for transformation* $\overset{T}{\Rightarrow}$ *is called* complete *for a set of labels* $M \subseteq \mathcal{L}$*, if any concrete reduction sequence of the form* $a \xleftarrow{sr,l_n} e_n \xleftarrow{sr,l_{n-1}} \dots \xleftarrow{sr,l_1} e_1 \overset{T}{\Rightarrow} e_0$ *where* $a \in \mathcal{A}$*,* $n > 0$*, and* $l_i \in \mathcal{L}$ *is rewritable by the CRSRS* $(\mathsf{cRS}, \xrightarrow{\mathcal{I}_M(\mathcal{J}_M(DF(\overset{T}{\Rightarrow})))})$*.*

In ARSs the label τ is used to represent standard reductions which are not explicitly mentioned in the diagrams, i.e. $\xleftarrow{sr,\tau}$ is interpreted as $\bigcup_{i=1}^{m} \xleftarrow{sr,l_i}$ where $l_1, \dots, l_m$ are the labels of $\mathcal{L}$ that do not occur in the general forking diagram.

Now that forking diagrams and their semantics are defined there are two further tasks: (i) We have also to deal with reductions $a \overset{T}{\Rightarrow} \ldots$, and (ii) diagrams may use several transformations $\overset{T_i}{\Rightarrow}$ in S_R. Thus for (i) we introduce answer diagrams, and for (ii) we will join forking diagrams of a set of transformations.

Definition 3.13. *An* answer diagram *for transformation $\overset{T}{\Rightarrow}$ is a rewrite rule of the form $A \overset{T}{\to} \rightsquigarrow S$ where S is a τ-free cARS. A* simple answer diagram *is defined analogously where τ-labels in S are allowed, but S is a simple cARS.*

The interpretation of a simple answer diagram w.r.t. a set $M \subseteq \mathcal{L}$ is

$$\mathcal{I}_M(A \overset{T}{\to} \rightsquigarrow S) := \{a_1 \bowtie e \rightsquigarrow a_2 \bowtie' e \mid a_1 \bowtie e \in \mathcal{I}_M(A \overset{T}{\to}), a_2 \bowtie' e \in \mathcal{I}_M(S)\}$$

We extend $\mathcal{I}_M$ to sets of simple answer diagrams joining the resulting sets.

For an answer diagram the set of simple answer diagrams w.r.t. a set of labels $M \subseteq \mathcal{L}$ is computed by the function $\mathcal{J}_M$ which is defined as follows:

$$\mathcal{J}_M(A \overset{T}{\to} \rightsquigarrow S) = \left\{ A \overset{T}{\to} \rightsquigarrow S' \;\middle|\; \begin{array}{l} S' \in \mathit{Exp}_\pi(V_{\neg M}(S)),\ \mathit{Exp}_\pi \text{ is an expansion,} \\ V_{\neg M} \text{ is a variable interpretation for } M \end{array} \right\}$$

We extend $\mathcal{J}_M$ to sets of answer diagrams such that the resulting sets are joined.

A set of answer diagrams $DA(\overset{T}{\Rightarrow})$ for transformation $\overset{T}{\Rightarrow}$ is complete w.r.t. a set of labels M *iff the set $\mathcal{I}_M(\mathcal{J}_M(DA(\overset{T}{\Rightarrow})))$ contains a rewrite rule with left hand side matching any possible cRS $a \overset{id}{\to} a \overset{T}{\Rightarrow} e$ for $a \in \mathcal{A}$ and $e \in \mathcal{E}$.*

Note that for an answer-preserving transformation $\overset{T}{\Rightarrow}$ a complete set of answer diagrams is $\{A \overset{T}{\to} \rightsquigarrow A\}$ and for a weakly answer-preserving transformation a complete set of answer diagrams can be constructed such that all answer diagrams are of the form $A \overset{T}{\to} \rightsquigarrow AI_m \ldots I_1$ where every I_j is an sr-symbol.

Definition 3.14. *Let $DA(\overset{T_1}{\Rightarrow}), \ldots, DA(\overset{T_n}{\Rightarrow})$ be sets of answer diagrams and $DF(\overset{T_1}{\Rightarrow}), \ldots, DF(\overset{T_n}{\Rightarrow})$ be sets of general forking diagrams. Let M be all labels of $\mathcal{L}$ that do not occur in any of the diagrams. The union D of these sets of diagrams is called* complete *for transformations $\overset{T_1}{\Rightarrow}, \ldots, \overset{T_n}{\Rightarrow}$ iff every set $DA(\overset{T_i}{\Rightarrow})$ is complete for $\overset{T_i}{\Rightarrow}$ w.r.t. M, every set $DF(\overset{T_i}{\Rightarrow})$ is complete for $\overset{T_i}{\Rightarrow}$ w.r.t. M, and the only transformations occurring in the diagrams are $\overset{T_1}{\to}, \ldots, \overset{T_n}{\to}$.*

We write $\mathcal{I}(D)$ instead of $\mathcal{I}_M(D)$ ($\mathcal{I}(\mathcal{J}(D))$ instead of $\mathcal{I}_M(\mathcal{J}_M(D))$, resp.) for a complete set D, since the set M is fixed by the completeness definition.

3.3 Proving Convergence Preservation

Definition 3.15. *A string rewriting system $(O, \rightharpoonup)$ is* leftmost terminating, *iff it is terminating w.r.t. the* leftmost rewriting relation $\rightharpoonup_l$*:*
$S_1 I_n \ldots I_1 S_2 \rightharpoonup_l S_1 S_R S_2$ iff $I_n \ldots I_1 \rightharpoonup S_R$ and $S_1 I_n \ldots I_2$ is $\rightharpoonup$-irreducible.

Proposition 3.16. *Let* $D = \bigcup_{i=1}^{n} DA(\overset{T_i}{\Rightarrow}) \cup \bigcup_{i=1}^{n} DF(\overset{T_i}{\Rightarrow})$ *be complete for* $\overset{T_1}{\Rightarrow}, \ldots, \overset{T_n}{\Rightarrow}$*. If the SRSARS* $(\mathsf{cARS}(\mathcal{J}(D)), \xrightarrow{\mathcal{J}(D)})$ *is (leftmost) terminating then the CRSRS* $(\mathsf{cRS}, \xrightarrow{\mathcal{I}(\mathcal{J}(D))})$ *is (leftmost) terminating.*

Proof. For termination, the claim holds, since a nonterminating rewriting sequence for the CRSRS $(\mathsf{cRS}, \xrightarrow{\mathcal{I}(\mathcal{J}(D))})$ can easily be transfered into a nonterminating rewriting sequence of the SRSARS $(\mathsf{cARS}(\mathcal{J}(D)), \xrightarrow{\mathcal{J}(D)})$. For leftmost termination the claim also holds, since completeness of D implies that always the leftmost transformation step is rewritten by the CRSRS, and there is a corresponding rewrite rule in $\mathcal{J}(D)$ which must be leftmost, since all left hand sides of rules in $\mathcal{J}(D)$ are of the form $\xleftarrow{sr,l_n} \ldots \xleftarrow{sr,l_1} \xrightarrow{T_j}$.

Proposition 3.17. *Let* $D = \bigcup_{i=1}^{n} DA(\overset{T_i}{\Rightarrow}) \cup \bigcup_{i=1}^{n} DF(\overset{T_i}{\Rightarrow})$ *be complete for* $\overset{T_1}{\Rightarrow}, \ldots, \overset{T_n}{\Rightarrow}$*. Let the CRSRS* $(\mathsf{cRS}, \xrightarrow{\mathcal{I}(\mathcal{J}(D))})$ *be terminating (leftmost terminating, resp.). Then the transformations* $\overset{T_1}{\Rightarrow}, \ldots, \overset{T_n}{\Rightarrow}$ *are convergence-preserving.*

Proof. Let $e_1 \overset{T_i}{\Rightarrow} e_0$ where $e_1{\Downarrow}$. Let $a \overset{id}{=} a \xleftarrow{sr,l_n} e_n \xleftarrow{sr,l_{n-1}} \ldots \xleftarrow{srl_1} e_1$ be a cRS witnessing $e_1{\Downarrow}$. We compute a normal form of the cRS $a \overset{id}{=} a \xleftarrow{sr,l_n} e_n \xleftarrow{sr,l_{n-1}} \ldots \xleftarrow{srl_1} e_1 \overset{T_i}{\Rightarrow} e_0$ using leftmost rewriting of the CRSRS $(\mathsf{cRS}, \xrightarrow{\mathcal{I}(\mathcal{J}(D))})$. We only have to argue that this normal form is of the form $a' \overset{id}{=} a' \xleftarrow{sr,l'_m} e'_m \ldots \xleftarrow{sr,l'_m} e_0$, which implies $e_0{\Downarrow}$. The definition of forking and answer diagrams and the completeness conditions imply that any rewrite step $\xrightarrow{\mathcal{I}(\mathcal{J}(D))}$ transforms a cRS into a cRS where the contained reductions are $\xleftarrow{sr,l}$-reductions and $\overset{T_j}{\Rightarrow}$-transformations. Completeness of the diagrams ensures that the reduction sequence is modifiable by $\xrightarrow{\mathcal{I}(\mathcal{J}(D))}$ as long as $\overset{T_j}{\Rightarrow}$-transformations are contained in the sequences. □

In general the other direction does not hold, since the SRSARS may be nonterminating, while the CRSRS is terminating. Propositions 3.16 and 3.17 imply:

Theorem 3.18. *Let* $D = \bigcup_{i=1}^{n} DA(\overset{T_i}{\Rightarrow}) \cup \bigcup_{i=1}^{n} DF(\overset{T_i}{\Rightarrow})$ *be complete for* $\overset{T_1}{\Rightarrow}, \ldots, \overset{T_n}{\Rightarrow}$*, and let the SRSARS* $(\mathsf{cARS}(\mathcal{J}(D)), \xrightarrow{\mathcal{J}(D)})$ *be (leftmost) terminating. Then the transformations* $\overset{T_1}{\Rightarrow}, \ldots, \overset{T_n}{\Rightarrow}$ *are convergence-preserving.*

3.4 A Rewriting System with Finitely Many Rules

A naive approach that encodes general diagrams with transitive closures adds rules $\xrightarrow{T,+} \rightsquigarrow \xrightarrow{T} \xrightarrow{T,+}$ and $\xrightarrow{T,+} \rightsquigarrow \xrightarrow{T}$ for any symbol $\xrightarrow{T,+}$. However, this is useless, since it it leads to nontermination. Hence, we provide another encoding which is suitable for automation. It translates +-symbols as nondeterministic rules using natural numbers to avoid nontermination. The translation is a little bit complex, since it has to respect the leftmost rewriting and it treats +-symbols in left hand sides and right hand sides differently.

Definition 3.19. *A (converging, resp.)* abstract reduction sequence with natural numbers *(NARS) (or cNARS, resp.) is a sequence* $I_n \dots I_1$ *(*$AI_n \dots I_1$*, resp.) where* A *represents any answer and each* I_j *is a symbol of the form* $\xleftarrow{sr,l}, \xrightarrow{T_i}, \langle w \rangle, \langle w, k \rangle, \langle w, \overline{k}+1 \rangle$ *where* $l \in \mathcal{L} \cup \{\tau\}$*, where* $w \in W$ *for a set of names* W *with* $W \cap (\mathcal{L} \cup \{\tau\}) = \emptyset$*, and* k *is either a natural number* ($k \in \mathbb{N}$) *or a* number variable*, i.e. a variable that may only be instantiated by natural numbers, and* $\overline{k}$ *is always a number variable. A NARS (cNARS, resp.) is called* ground *iff it does not contain number variables.*

Definition 3.20. *A* number substitution σ *assigns a natural number to any number variable. The extension of* σ *to NARS-symbols is the identity except for the cases* $\sigma(\langle w, k \rangle) = \langle w, \sigma(k) \rangle$*,* $\sigma(\langle w, k+1 \rangle) = \langle w, k' \rangle$*, where* k *is a number variable, and* $k' = \sigma(k) + 1 \in \mathbb{N}$*.*

We now define rewriting on ground cNARSs.

Definition 3.21. *Let* D *be a set of rules of the form* $S_L \rightsquigarrow S_R$ *where* S_L, S_R *are NARSs. Let* $\mathsf{gcNARS}(D)$ *be the set of all ground cNARS that can be built by instantiating the symbols occurring in* D *by any number substitution* σ*. The rewriting system (*$\mathsf{gcNARS}(D), \xrightarrow{D}$*) is called an* encoded rewrite system on abstract reduction sequence *(ERSARS) where* $\xrightarrow{D}$ *is defined by: If* $S = S' S'_L S''$*,* $S_L \rightsquigarrow S_R \in D$*,* σ *is a number substitution with* $\sigma(S_L) = S'_L$*, then* $S \xrightarrow{D} S' \sigma(S_R) S''$*.*

Definition 3.22. *For a general forking or answer diagram* $S_L \rightsquigarrow S_R$ *and* $M \subseteq \mathcal{L}$ *the translation* $\mathcal{V}_M$ *is a finite set of rewrite rules over ground ARSs:*

$$\mathcal{V}_M(S_L \rightsquigarrow S_R) := \bigcup \{V_{\neg M}(S_L) \rightsquigarrow V_{\neg M}(S_R) \mid V_{\neg M} \text{ is a variable interpretation}\}$$

Given a set $D = \bigcup_i \{S_{i,L} \rightsquigarrow S_{i,R}\}$ *of general forking and general answer diagrams and* $M \subseteq \mathcal{L}$ *the translation* $\mathcal{K}_M(D)$ *is defined as follows:*

First all (usual) variables are interpreted, resulting in the set $D' := \bigcup_i \{\mathcal{V}_M(S_{i,L} \rightsquigarrow S_{i,R})\}$*. For every rule* $S_L \rightsquigarrow S_R \in D'$ *the set* $\mathcal{K}_M(D)$ *contains a rule* $\mathcal{K}_L(S_L) \to \mathcal{K}_R(S_R)$ *perhaps together with some further rules.*

Construction of $\mathcal{K}_L(S_L)$*: Let* $S_L = I_n \dots I_1 \xrightarrow{T_i}$ *where* I_j *is either* $\xleftarrow{sr,l_j}$*, or* $\xleftarrow{sr,l_j,+}$*, or (for* $j = 1$*)* $I_j = A$*. Let* $K_j := \langle w_j \rangle$ *if* $I_j = \xleftarrow{sr,l_j,+}$ *and* $K_j := I_j$ *otherwise, where* $w_j \in W$ *are fresh names (chosen fresh for any new rule). Then we set* $\mathcal{K}_L(S_L) := K_n \dots K_1 \xrightarrow{T_i}$*. For any* I_j *which is of the form* $\xleftarrow{sr,l_j,+}$ *we add two so-called* contraction rules*:* $\xleftarrow{sr,l_j} K_{j-1} \dots K_1 \xrightarrow{T_i} \rightsquigarrow K_j K_{j-1} \dots K_1 \xrightarrow{T_i}$ *and* $\xleftarrow{sr,l_j} K_j K_{j-1} \dots K_1 \xrightarrow{T_i} \rightsquigarrow K_j K_{j-1} \dots K_1 \xrightarrow{T_i}$*.*

Construction of $\mathcal{K}_R(S_R)$**:** *Let* $S_R = I_n \dots I_1$*. If none of the* I_j *contains a* $+$ *then the translation is* $\mathcal{K}_R(S_R) := I_n \dots I_1$*. Otherwise there is at least one* $+$*. Let* $L_j := \xrightarrow{T_i}$ *if* $I_j = \xrightarrow{T_i,+}$*,* $L_j := \xleftarrow{sr,l_j}$ *if* $I_j = \xleftarrow{sr,l_j,+}$ *and* $L_j := I_j$ *otherwise.*

Let $w'_j \in W$ *(for* $j \in \{1, \dots, n\}$*) be fresh names. Let* I_a *be the rightmost* I_j *that contains a* $+$*, then we set* $\mathcal{K}_R(S_R) := \langle w'_a, k \rangle L_{a-1} \dots L_1$ *where* k *is a number variable. For all* I_j *with* $I_j = \xrightarrow{T_i,+}$ *or* $I_j = \xleftarrow{sr,l_j,+}$ *we additionally add so-called*

expansion rules $\langle w'_j, k+1\rangle \rightsquigarrow \langle w'_j, k\rangle L_j$ *and* $\langle w'_j, 1\rangle \rightsquigarrow L_n \dots L_j$. *If there exists* $m > j$ *where* I_m *contains a* $+$, *then for the smallest such* m *we also add the expansion rule* $\langle w'_j, k+1\rangle \rightsquigarrow \langle w'_m, k\rangle L_{m-1} \dots L_j$.

For complete sets of diagrams, M is the set of labels that do not occur in any of the diagrams. In this case we omit the index M in $\mathcal{K}_M$.

The symbols $\langle w_i\rangle$ and $\langle w'_j, k\rangle$ together with the additional rules are used to interpret the transitive closure symbols on the left and the right hand side of rules in forking and answer diagrams. It is easy to verify that any rewriting sequence using only the contraction rules must be finite, and also that any rewriting sequence using only the expansions rules is also finite.

Example 3.23. Let D be the set consisting of the third diagram from Example 3.6. For $\mathcal{L} = \{lll, llet, seq, \dots\}, M = \mathcal{L} \setminus \{lll, llet\}$ the translation $\mathcal{K}_M(D)$ is: $\{\langle w\rangle \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \langle w', k\rangle,\ \xleftarrow{n,lll} \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \langle w\rangle \xrightarrow{i\mathcal{S},llet},\ \xleftarrow{n,lll} \langle w\rangle \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \langle w\rangle \xrightarrow{i\mathcal{S},llet},$ $\langle w', k+1\rangle \rightsquigarrow \langle w', k\rangle \xleftarrow{n,lll},\ \langle w', 1\rangle \rightsquigarrow \xleftarrow{n,lll}\}$

Lemma 3.24. *Let* $D = \bigcup_{i=1}^{n} DA(\overset{T_i}{\Longrightarrow}) \cup \bigcup_{i=1}^{n} DF(\overset{T_i}{\Longrightarrow})$ *be complete for* $\overset{T_1}{\Longrightarrow}, \dots, \overset{T_n}{\Longrightarrow}$ *and* $S_L \rightsquigarrow S_R \in \mathcal{J}(D)$. *Then* $S_L \xrightarrow{\mathcal{K}(D),*}_l S_R$.

Proof. Since $S_L \rightsquigarrow S_R \in \mathcal{J}(D)$ there are expansions $Exp_\pi, Exp_{\pi'}$ and a variable interpretation $V_{\neg M}$ such that $S_L = Exp_\pi(V_{\neg M}(S'_L))$ and $S_R = Exp_{\pi'}(V_{\neg M}(S'_R))$ where $S'_L \rightsquigarrow S'_R \in D$. Let $S'_L = I'_n \dots I'_1 \xrightarrow{T_i}$, i.e. $S_L = Exp_{\pi(1)}(V_{\neg M}(I'_n)) \dots Exp_{\pi(n)}(V_{\neg M}(I'_1)) \xrightarrow{T_i}$. Let $K_j := \langle w_j\rangle$ if $V_{\neg M}(I'_j)$ contains a $+$ and $K_j := V_{\neg M}(I'_j)$ otherwise. Using the contraction rules introduced by $\mathcal{K}(D)$ we rewrite $S_L = Exp_{\pi(1)}(V_{\neg M}(I'_n)) \dots Exp_{\pi(n)}(V_{\neg M}(I'_1)) \xrightarrow{T_i} \xrightarrow{\mathcal{K}(D),*}_l$ $Exp_{\pi(1)}(V_{\neg M}(I'_n)) \dots K_1 \xrightarrow{T_i} \xrightarrow{\mathcal{K}(D),*}_l \dots \xrightarrow{\mathcal{K}(D),*}_l K_n \dots K_1 \xrightarrow{T_i} = \mathcal{K}(V_{\neg M}(S'_L))$. All these steps are leftmost, since the rightmost symbol $\xrightarrow{T_i}$ is always part of the redex and always kept. Now we apply the rule $\mathcal{K}(V_{\neg M}(S'_L)) \xrightarrow{\mathcal{K}(D)}_l \mathcal{K}(V_{\neg M}(S'_R))$ (which is again leftmost) and have to show that $\mathcal{K}(V_{\neg M}(S'_R))$ can be rewritten into S_R by leftmost rewriting using $\xrightarrow{\mathcal{K}(D)}$.

If S_R does not contain a $+$-symbol, then this is obvious. Suppose that S_R contains at least one $+$-symbol. Let $S'_R = J'_m \dots J'_1$, i.e. $S_R = Exp_{\pi'(1)}(V_{\neg M}(J'_m)) \dots Exp_{\pi'(m)}(V_{\neg M}(J'_1))$ and let $L_j = \xrightarrow{T_i}$ if $V_{\neg M}(J'_j) = \xrightarrow{T_i,+}$, $L_j = \xleftarrow{sr,l}$ if $V_{\neg M}(J'_j) = \xleftarrow{sr,l,+}$, and $L_j = V_{\neg M}(J'_j)$ otherwise. Moreover let us assume that $J'_{a_r}, \dots J'_{a_1}$ are the symbols that contain a $+$, such that for $i, j \in \{1, \dots, r\}$ with $i \neq j$ we have $a_i < a_j$. Then $S_R = Q_m \dots Q_1$ where every Q_j consists of k_j repetitions of L_j, i.e. it is a string of the form $L_j \dots L_j$. Let $s := \sum_{i \in \{a_1, \dots, a_k\}} k_i$. We choose this number s during the rewriting step $\mathcal{K}(V_{\neg M}(S'_L)) \xrightarrow{\mathcal{K}(D)}_l \mathcal{K}(V_{\neg M}(S'_R))$, and then iteratively build the string S_R using the expansion rules introduced by $\mathcal{K}(D)$: $\mathcal{K}(V_{\neg M}(S'_L)) \xrightarrow{\mathcal{K}(D)}_l \mathcal{K}(V_{\neg M}(S'_R)) =$

$\langle w'_{a_1}, s\rangle Q_{a_1-1} \dots Q_1 \xrightarrow{\mathcal{K}(D),*}_l \langle w'_{a_2}, s - k_{a_1}\rangle Q_{a_2-1} \dots Q_1 \xrightarrow{\mathcal{K}(D),*}_l \dots \xrightarrow{\mathcal{K}(D),*}_l \langle w'_{a_r}, k_{a_r}\rangle Q_{a_r-1} \dots Q_1 \xrightarrow{\mathcal{K}(D),*}_l Q_m \dots Q_1 = S_R$. All steps are leftmost, since always the leftmost symbol is reduced (which is of the form $\langle w'_j, s'\rangle$). □

Proposition 3.25. *Let $D = \bigcup_{i=1}^n DA(\overset{T_i}{\Longrightarrow}) \cup \bigcup_{i=1}^n DF(\overset{T_i}{\Longrightarrow})$ be complete for $\overset{T_1}{\Longrightarrow}, \dots, \overset{T_n}{\Longrightarrow}$. Then leftmost termination of the ERSARS $(\mathsf{gcNARS}(\mathcal{K}(D)), \xrightarrow{\mathcal{K}(D)})$ implies leftmost termination of the SRSARS $(\mathsf{cARS}(\mathcal{J}(D)), \xrightarrow{\mathcal{J}(D)})$.*

Proof. We show that a leftmost diverging rewriting sequence of the SRSARS can be transformed to a leftmost diverging rewriting sequence of the ERSARS. Assume there is a diverging reduction. We consider a single step $S_1 S_L S_2 \xrightarrow{\mathcal{J}(D)}_l S_1 S_R S_2$ from this diverging reduction. Then S_L must be of the form $I_1 \dots I_n \xrightarrow{T_i}$ where all I_k are sr-symbols or A. If S_1 does not contain a transformation-symbol, then Lemma 3.24 implies that $S_1 S_L S_2 \xrightarrow{\mathcal{K}(D),*}_l S_1 S_R S_2$: The rewriting step must be leftmost, since at the beginning a transformation step is required on the end of the redex, and since the rewriting generates S_R from right to left.

We now consider the case that S_1 contains other transformation symbols, w.l.o.g. let $S_1 = S_3 \xrightarrow{T_j} S_4$ such that S_4 does not contain transformation-symbols. Then perhaps there are some leftmost rewriting steps possible inside $S_3 \xrightarrow{T_j}$ using $\xrightarrow{\mathcal{K}(D),*}_l$: These can only be steps using the contraction rules. Since contraction rules cannot remove the rightmost transformation-symbol in the redex and since they are terminating, the following rewriting sequence is possible $S_3 \xrightarrow{T_j} S_4 S_L S_2 \xrightarrow{\mathcal{K}(D),*}_l S'_3 \xrightarrow{T_j} S_4 S_L S_2 \xrightarrow{\mathcal{K}(D),*}_l S'_3 \xrightarrow{T_j} S_4 S_R S_2$. Any rewriting sequence of $\xrightarrow{\mathcal{K}(D)}_l$ now cannot modify the prefix $S'_3 \xrightarrow{T_j}$. Moreover, all rewriting steps of $\xrightarrow{\mathcal{J}(D)}_l$ starting with $S_3 \xrightarrow{T_j} S_4 S_R S_2$ also do not modify the prefix $S_3 \xrightarrow{T_j}$ and thus it does not make a difference if we replace S_3 by S'_3. □

Proposition 3.25 and Theorem 3.18 imply:

Theorem 3.26. *Let $D = \bigcup_{i=1}^n DA(\overset{T_i}{\Longrightarrow}) \cup \bigcup_{i=1}^n DF(\overset{T_i}{\Longrightarrow})$ be complete for $\overset{T_1}{\Longrightarrow}, \dots, \overset{T_n}{\Longrightarrow}$. Then leftmost termination of the ERSARS $(\mathsf{gcNARS}(\mathcal{K}(D)), \xrightarrow{\mathcal{K}(D)})$ implies that all transformations $\overset{T_i}{\Longrightarrow}$ are convergence preserving.*

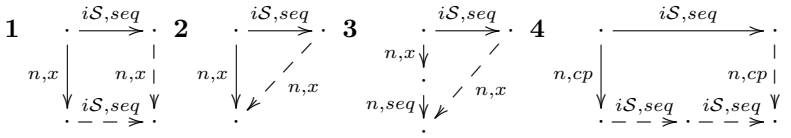
Theorem 3.27. *Let $\overset{T_1}{\Longrightarrow}$ be CP-sufficient for $\overset{T}{\Rightarrow}$ and let $\overset{T'_1}{\Longleftarrow}$ be CP-sufficient for $\overset{T}{\Longleftarrow}$ and let $D = \bigcup_{i=1}^n DA(\overset{T_i}{\Longrightarrow}) \cup \bigcup_{i=1}^n DF(\overset{T_i}{\Longrightarrow})$ be complete for $\overset{T_1}{\Longrightarrow}, \dots, \overset{T_n}{\Longrightarrow}$, $D' = \bigcup_{i=1}^m DA(\overset{T'_i}{\Longleftarrow}) \cup \bigcup_{i=1}^m DF(\overset{T'_i}{\Longleftarrow})$ be complete for $\overset{T'_1}{\Longleftarrow}, \dots, \overset{T'_m}{\Longleftarrow}$, such that both ERSARSs $(\mathsf{gcNARS}(\mathcal{K}_M(D)), \xrightarrow{\mathcal{K}_M(D)})$ and $(\mathsf{gcNARS}(\mathcal{K}_M(D')), \xrightarrow{\mathcal{K}_M(D')})$ are (leftmost) terminating. Then $\overset{T}{\Rightarrow}$ is a correct program transformation.*

Proof. Theorem 3.26 shows that $\overset{T_1}{\Longrightarrow}$ and $\overset{T'_1}{\Longleftarrow}$ are convergence preserving and thus CP-sufficiency shows that $\overset{T}{\Rightarrow}$ is a correct program transformation.

4 Encoding ARSs and Sets of Diagrams as ITRSs

For the automation of correctness proofs we left open how to check for leftmost termination of an ERSARS derived by complete sets of forking and answer diagrams according to Theorems 3.27.

If the diagrams do not contain transitive closures, then the ERSARS is also an SRSARS with finitely many rules. In this case the SRSARS can be encoded as a term rewriting system: A step $\xrightarrow{T_i}$ is encoded as 1-ary function symbol ti, a step $\xleftarrow{sr,l_i}$ is encoded as a 1-ary function symbol $srli$, and the answer token A is encoded as a constant A. The string rewriting rules are translated into term rewriting rules, where left and right hand sides are both encoded from *right to left*, e.g. the rule $\xleftarrow{sr,l_1}\xrightarrow{T_1} \rightsquigarrow \xrightarrow{T_2}\xleftarrow{sr,l_2}$ is encoded as the term rewriting rule $t1(srl1(X)) \to srl2(t1(X))$ where X is a variable. It is easy to verify that leftmost termination of the SRSARS is implied by innermost termination of the TRS. We illustrate this encoding by an example from [21] for the calculus LR (see Example 2.2). We consider the transformation $\overset{seq}{\Longrightarrow}$, which is used for sequentialization, and reduces an expression `seq` e_1 e_2 to e_2 if e_1 is a value or bound to a value. The complete set of general forking diagrams $DF(\overset{i\mathcal{S},seq}{\Longrightarrow})$ for the transformation $\overset{i\mathcal{S},seq}{\Longrightarrow}$, which is CP-sufficient for the transformation $\overset{seq}{\Longrightarrow}$, is:



Since transformation $\overset{i\mathcal{S},seq}{\Longrightarrow}$ is answer-preserving, the answer diagrams are $DA(\overset{i\mathcal{S},seq}{\Longrightarrow}) = \{A \xrightarrow{i\mathcal{S},seq} \rightsquigarrow A\}$. The encoding of the corresponding SRSARS $\mathcal{J}(DF(\overset{i\mathcal{S},seq}{\Longrightarrow}) \cup DA(\overset{i\mathcal{S},seq}{\Longrightarrow}))$ as a TRS is as follows, where X denotes a term-variable, and all other symbols are function symbols.

1 $iSseq(ntau(X)) \to ntau(iSseq(X))$
$iSseq(nseq(X)) \to nseq(iSseq(X))$
$iSseq(ncp(X)) \to ncp(iSseq(X))$

2 $iSseq(ntau(X)) \to ntau(X)$
$iSseq(nseq(X)) \to nseq(X)$
$iSseq(ncp(X)) \to ncp(X)$

3 $iSseq(ntau(nseq(X))) \to ntau(X)$
$iSseq(nseq(nseq(X))) \to nseq(X)$
$iSseq(ncp(nseq(X))) \to ncp(X)$

4 $iSseq(ncp(X)) \to ncp(iSseq(iSseq(X)))$

Answer diagram: $iSseq(A) \to A$

For $\overset{i\mathcal{S},seq}{\Longrightarrow}$, and also for its inverse $\overset{i\mathcal{S},seq}{\Longleftarrow}$ innermost termination of the encoded complete diagram sets could be automatically shown via AProVE. Hence by Theorem 3.18 we can conclude correctness of the transformation.

If transitive closures occur on right hand sides of the diagrams, then an encoding into a usual TRS is not possible, since the corresponding rule in the ERSARS introduces a free number variable (which is then used for the expansion of the transitive closures). However, *conditional integer term rewriting systems* (ITRSs) is a formalism that fits for encoding ERSARS, since they allow free variables on right hand sides of rules which may only

be instantiated by normal forms during rewriting. Also integers as well as conditions including arithmetic operations, comparison of integers, and Boolean connectives are already present in ITRSs (see e.g., [4]). Moreover, innermost termination of ITRSs can also be treated by the automated termination prover AProVE [5,4]. Since innermost termination of the encoded ITRSs then implies leftmost termination of an ERSARS we can use AProVE to show correctness of program transformations. The translation is as before, where the introduced names w_i in contraction rules are encoded as 1-ary function symbols, the names w'_i in expansion rules are encoded as 2-ary function symbols, natural numbers are represented by integers, and number variables are represented by variables together with constraints. Example 3.6 shows a complete set of forking diagrams for the transformation $\stackrel{i\mathcal{S},llet}{\Longrightarrow}$, and Example 3.23 shows the encoding of the third diagram as an ERSARS. An encoding of these rules as an ITRS is as follows where X, K are variables and all other symbols are function symbols:

$$iSllet(w(X)) \to v(K,X) \text{ with } K > 0$$
$$iSllet(w(nlll(X))) \to iSllet(w(X)) \qquad iSllet(nlll(X)) \to iSllet(w(X))$$
$$v(K,X) \to nlll(v(K-1,X)) \text{ if } K > 1 \qquad v(1,X) \to nlll(X)$$

The first rule encodes the diagram, the other rules are contraction rules (using the function symbol w), and expansion rules (using the function symbol v). The first constraint $K > 0$ ensures that a positive integer is chosen, and the constraint $K > 1$ ensures that K is a positive integer after rewriting. Innermost termination of the ITRS-encoded complete sets of forking and answer diagrams for $\stackrel{i\mathcal{S},llet}{\Longrightarrow}$ and for $\stackrel{i\mathcal{S},llet}{\Longleftarrow}$ can be checked using AProVE. This implies leftmost termination of the corresponding ERSARSs and thus by Theorem 3.27 correctness of the transformation *llet* is shown automatically.

We encoded complete sets of diagrams for several program transformations from [21] and they could all be shown as innermost terminating using AProvE. The encoded diagrams and the termination proofs can be found on our website[2].

5 Conclusion

Future work is to connect the automated termination prover with the diagram calculator of [16,17] and thus to complete the tool for automated correctness proofs of program transformations. Another direction is to check more sets of diagrams which may require more sophisticated encoding techniques.

Acknowledgments We thank Carsten Fuhs for pointing us to ITRS and his support on AProVE and the anonymous reviewers for their valuable comments.

References

1. H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, New York, 1984.

[2] `http://www.ki.cs.uni-frankfurt.de/research/dfg-diagram/auto-induct/`

2. E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated certified proofs with CiME3. In M. Schmidt-Schauß (ed.), *RTA 22*, *LIPIcs* 10, pp. 21–30. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
3. J. Ford and I. A. Mason. Formal foundations of operational semantics. *Higher Order Symbol. Comput.*, 16(3):161–202, 2003.
4. C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In R. Treinen (ed.), *RTA 20*, *LNCS* 5595, pp. 32–47. Springer, 2009.
5. J. Giesl, P. Schneider-Kamp, and R. Thiemann. Automatic termination proofs in the dependency pair framework. In U. Furbach and N. Shankar (eds.), *3rd IJCAR*, *LNCS* 4130, pp. 281–286. Springer, 2006.
6. M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean termination tool 2. In R. Treinen (ed.) *20th RTA*, *LNCS* 5595, pp. 295–304. Springer, 2009.
7. A. Kutzner and M. Schmidt-Schauß. A nondeterministic call-by-need lambda calculus. In M. Felleisen, P. Hudak, and C. Queinnec (eds.), *3rd ICFP*, pp. 324–335. ACM , 1998.
8. E. Machkasova and F. A. Turbak. A calculus for link-time compilation. In G. Smolka (ed.), *9th ESOP*, *LNCS* 1782, pp. 260–274. Springer, 2000.
9. M. Mann. Congruence of bisimulation in a non-deterministic call-by-need lambda calculus. *Electron. Notes Theor. Comput. Sci.*, 128(1):81–101, 2005.
10. J.H. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT, 1968.
11. J. Niehren, D. Sabel, M. Schmidt-Schauß, and J. Schwinghammer. Observational semantics for a concurrent lambda calculus with reference cells and futures. *Electron. Notes Theor. Comput. Sci.*, 173:313–337, 2007.
12. J. Niehren, J. Schwinghammer, and G. Smolka. A concurrent lambda calculus with futures. *Theoret. Comput. Sci.*, 364(3):338–356, 2006.
13. S. Peyton Jones. *Haskell 98 language and libraries: the Revised Report*. Cambridge University Press, 2003.
14. S. Peyton Jones, A. Gordon, and S. Finne. Concurrent Haskell. In G. Steele (ed.) *23th POPL*, pp. 295–308. ACM, 1996.
15. G. D. Plotkin. Call-by-name, call-by-value, and the lambda-calculus. *Theoret. Comput. Sci.*, 1:125–159, 1975.
16. C. Rau and M. Schmidt-Schauß. Towards correctness of program transformations through unification and critical pair computation. In M. Fernandez (ed.), *24th UNIF*, *EPTCS* 42, pp. 39–54, 2010.
17. C. Rau and M. Schmidt-Schauß. A unification algorithm to compute overlaps in a call-by-need lambda-calculus with variable-binding chains. In F. Baader, B. Morawska, and J. Otop (eds.), *25th UNIF*, pp. 35–41, 2011.
18. D. Sabel and M. Schmidt-Schauß. A call-by-need lambda-calculus with locally bottom-avoiding choice: Context lemma and correctness of transformations. *Math. Structures Comput. Sci.*, 18(03):501–553, 2008.
19. D. Sabel and M. Schmidt-Schauß. A contextual semantics for Concurrent Haskell with futures. In M. Hanus (ed.), *13th PPDP*, pp. 101–112, ACM, 2011.
20. M. Schmidt-Schauß and D. Sabel. On generic context lemmas for higher-order calculi with sharing. *Theoret. Comput. Sci.*, 411(11-13):1521 – 1541, 2010.
21. M. Schmidt-Schauß, M. Schütz, and D. Sabel. Safety of Nöcker's strictness analysis. *J. Funct. Programming*, 18(04):503–551, 2008.
22. J. B. Wells, D. Plump, and F. Kamareddine. Diagrams for meaning preservation. In R. Nieuwenhuis (ed.), *14th RTA*, *LNCS* 2706, pp. 88 –106, Springer, 2003.