

# Narrowing Failure in Functional Logic Programming, Extended Version

F. J. López-Fraguas and J. Sánchez-Hernández

Dep. Sistemas Informáticos y Programación, Univ. Complutense de Madrid  
{fraguas,jaime}@sip.ucm.es

**Abstract.** Negation as failure is an important language feature within the logic programming paradigm. The natural notion generalizing negation as failure in a functional logic setting is that of finite failure of reduction. In previous works we have shown the interest of using such programming construct when writing functional logic programs, and we have given a logical status to failure by means of proof calculi designed to deduce failures from programs. In this paper we address the problem of the operational mechanism for the execution of functional logic programs using failure. Our main contribution is the proposal of a narrowing relation able to deal with failures, which is constructive in the usual sense of the term in the context of negation, that is, narrowing is able to find substitutions for variables even in presence of failures. As main technical results, we prove correctness and completeness of the narrowing relation with respect to the proof-theoretic semantics.

## 1 Introduction

**Motivation** Functional logic programming (*FLP* for short) tries to combine the nicest properties of functional and logic programming (see [9] for a now ‘classical’ survey on *FLP*). Theoretical aspects of *FLP* are well established (e.g. [8]) and there are also practical implementations such as *Curry* [10] or *TOY* [1, 11]. Disregarding syntax, both pure Prolog and (a wide subset of) Haskell are subsumed by those systems. The usual claim is then that by the use of an *FLP* system one can choose the style of programming better suited to each occasion.

There is nevertheless a very important feature in the logic programming (*LP* for short) paradigm, namely *negation as failure* [6], yet not available in *FLP* systems. Negation is a major issue in the *LP* field, both at the theoretical and the practical levels. There are hundreds of papers about various aspects of negation in *LP* (see e.g. [4] for a survey), and almost all real Prolog programs of medium size use negation at some point.

This situation poses some problems to *FLP*, since logic programs using negation cannot be directly seen as *FLP* programs. This would be a not very serious inconvenience if other features of *FLP* could easily replace the use of failure. This happens in some cases, when the possibility of defining two-valued boolean functions is a good alternative to the use of negation in a logic program.

But there are problems where the use of failure as a programming construct is a real aid to the writing of concise declarative programs. We give an example – to be used several times throughout the paper – of that situation in the context of *FLP*. Some other examples can be found in [14].

**Example** Consider the problem of determining if there is a path connecting two nodes of an acyclic directed graph. A typical way of representing a graph in current *FLP* languages is by means of a non-deterministic function *next*, with rules of the form  $next(N) \rightarrow N'$ , indicating that there is an arc from  $N$  to  $N'$ . A concrete graph with nodes  $a$ ,  $b$ ,  $c$  and  $d$  could be given by the rules:

$$\begin{array}{ll} next(a) \rightarrow b & next(b) \rightarrow c \\ next(a) \rightarrow c & next(b) \rightarrow d \end{array}$$

And to determine if there is a path from  $X$  to  $Y$  we can define:

$$path(X, Y) \rightarrow \text{if } eq(X, Y) \text{ then true else } path(next(X), Y)$$

where *eq* stands for strict equality.

Notice that *path* behaves as a semidecision procedure recognizing only the positive cases, and there is no clear way (in ‘classical’ *FLP*) of completing its definition with the negatives ones, unless we change from the scratch the representation of graphs. Therefore we cannot, for instance, program in a direct way a property like

$$safe(X) \Leftrightarrow X \text{ is not connected with } d$$

To this purpose, something like negation as failure would be very useful. Since predicates can be programmed in *FLP* systems like *true*-valued functions, a natural *FLP* generalization of negation as failure is given by the notion of *failure of reduction to head normal form*. This could be expressed by means of a ‘primitive’ function *fails*, with the following intended behavior:

$$fails(e) ::= \begin{cases} true & \text{if } e \text{ fails to be reduced to hnf} \\ false & \text{otherwise} \end{cases}$$

Using such a primitive *fails* it is easy to define the property *safe*:

$$safe(X) \rightarrow fails(path(X, d))$$

With this definition and the previous graph, *safe(X)* becomes *true* for  $X = c$  and *false* for  $X = a$ ,  $X = b$  and  $X = d$ .

**Previous and related work. Aim of the paper** In some previous works [12, 14, 13] we addressed the problem of failure within *FLP*, from the point of view of its semantic foundations. Our starting point was *CRWL* [7, 8], a general framework for *FLP*, based on a Constructor based *ReWriting Logic*. From the point of view of programming, a fundamental notion of *CRWL* (as well as for existing *FLP* systems like *Curry* [10] or *TOY* [11, 1]) is that of non-strict non-deterministic function, for which *call-time choice semantics* is considered. Programs in *CRWL* have a logical semantics given by the logical consequences of the program according to a proof calculus able to prove reduction statements of the form  $e \rightarrow t$ , meaning that one of the possible reductions of an expression  $e$  results in the (possibly partial) value  $t$ .

Our first step [12, 14] for dealing with failure in *FLP*, was to extend the rewriting logic *CRWL* to *CRWLF* (*CRWL* “with failure”). The main insight was to replace statements  $e \rightarrow t$  by statements  $e \triangleleft \mathcal{C}$ , where  $\mathcal{C}$  are sets of partial values (called *Sufficient Approximation Sets* or *SAS*s) corresponding to the different possibilities

for reducing  $e$ . In [13] we realized the benefits of making more explicit the set nature of non deterministic functions, and therefore reformulated *CRWLF* by giving a set-oriented view of programs. This is done even at the syntactic level, by introducing classical mathematical set notation, like union or indexed unions; union is useful to transform programs to an inductively sequential format [2], and indexed unions turn to be useful for expressing call-time choice semantics and sharing.

One of the motivations for the set-oriented view adopted in [13] was our thought that such approach, by reflecting better the meaning of programs, would be an appropriate semantic basis upon which develop a suitable operational semantics for functional logic programs using failure. This is exactly the purpose of this paper. We propose an operational mechanism given by a narrowing relation which can operate with failure, that is, which is able to narrow expressions of the form  $fails(e)$ . Since we start from a precise semantic interpretation of failure, we are able to prove correctness and completeness of narrowing with respect to the semantics.

A major feature of our proposal is that the narrowing relation is *constructive* in the usual sense given to the term in the context of negation [5, 18, 19]: if an expression  $fails(e)$  contains variables, it can be still narrowed to obtain appropriate substitutions for the variables. For instance, in the example of the graph, our narrowing relation is able to narrow the expression  $safe(X)$  to obtain the value *true* together with the substitution  $X = c$  and the value *false* together with the substitutions (corresponding to different computations)  $X = a$ ,  $X = b$  and  $X = d$ .

There are very few works about negation in *FLP*. In [15] the work of Stuckey about constructive negation [18, 19] is adapted to the case of *FLP* with strict functions and innermost narrowing as operational mechanism. In [16] a similar work is done for the case of non-strict functions and lazy narrowing. This approach, apart from being in the technical side very different from ours, does not take into account non-determinism of functions, an essential aspect in our work.

The organization of the paper is as follows: Section 2 contains some technical preliminaries. In Section 3 we present (a slight variant of) the set-oriented semantic framework for failure of [13], including the proof calculus, together with some new results which are needed for subsequent sections. Section 4 is the core of the paper, where we define the narrowing relation and prove the results of correctness and completeness with respect the logical semantics. Finally, Section 5 summarizes some conclusions and hints for future work.

## 2 Technical Preliminaries

We assume a signature  $\Sigma = DC_{\Sigma} \cup FS_{\Sigma} \cup \{fails\}$  where  $DC_{\Sigma} = \bigcup_{n \in \mathbb{N}} DC_{\Sigma}^n$  is a set of *constructor* symbols containing at least the usual boolean ones *true* and *false*,  $FS_{\Sigma} = \bigcup_{n \in \mathbb{N}} FS_{\Sigma}^n$  is a set of *function* symbols, all of them with associated arity and such that  $DC_{\Sigma} \cap FS_{\Sigma} = \emptyset$ , and  $fails \notin DC \cup FS$  (with arity 1). We also assume a countable set  $\mathcal{V}$  of *variable* symbols. We write  $Term_{\Sigma}$  for the set of (total) *terms* (we say also *expressions*) built over  $\Sigma$  and  $\mathcal{V}$  in the usual way, and we distinguish the subset  $CTerm_{\Sigma}$  of (total) constructor terms or (total) *cterm*s, which only makes use of  $DC_{\Sigma}$  and  $\mathcal{V}$ . The subindex  $\Sigma$  will usually be omitted. Terms intend to represent possibly reducible expressions, while cterms represent data values, not further reducible.

The constant (0-arity constructor) symbol  $\text{F}$  is explicitly used in our terms, so we consider the signature  $\Sigma_{\text{F}} = \Sigma \cup \{\text{F}\}$ . This symbol  $\text{F}$  will be used to express failure of reduction. The sets  $Term_{\text{F}}$  and  $CTerm_{\text{F}}$  are defined in the natural way. The denotational semantics also uses the constant symbol  $\perp$ , that plays the role of the undefined value. We define  $\Sigma_{\perp, \text{F}} = \Sigma \cup \{\perp, \text{F}\}$ ; the sets  $Term_{\perp, \text{F}}$  and  $CTerm_{\perp, \text{F}}$  of (partial) terms and (partial) cterms respectively, are defined in a natural way. Partial cterms represent the result of partially evaluated expressions; thus, they can be seen as approximations to the value of expressions in the denotational semantics.

As usual notations we will write  $X, Y, Z, \dots$  for variables,  $c, d$  for constructor symbols,  $f, g$  for functions,  $e$  for terms and  $s, t$  for cterms. In all cases, primes ( $'$ ) and subindices can be used.

The sets of substitutions  $CSubst$ ,  $CSubst_{\text{F}}$  and  $CSubst_{\perp, \text{F}}$  are defined as applications from  $\mathcal{V}$  into  $CTerm$ ,  $CTerm_{\text{F}}$  and  $CTerm_{\perp, \text{F}}$  respectively. We will write  $\theta, \sigma, \mu$  for general substitutions and  $\epsilon$  for the identity substitution. The notation  $\theta\sigma$  stands for the usual composition of substitutions. All the considered substitutions are idempotent ( $\theta\theta = \theta$ ). We also write  $[X_1/t_1, \dots, X_n/t_n]$  for the substitution that maps  $X_1$  into  $t_1, \dots, X_n$  into  $t_n$ .

Given a set of constructor symbols  $D$ , we say that the terms  $t$  and  $t'$  have a *D-clash* if they have different constructor symbols of  $D$  at the same position. We say that two tuples of cterms  $t_1, \dots, t_n$  and  $t'_1, \dots, t'_n$  have a *D-clash* if for some  $i \in \{1, \dots, n\}$  the cterms  $t_i$  and  $t'_i$  have a *D-clash*.

A natural *approximation ordering*  $\sqsubseteq$  over  $Term_{\perp, \text{F}}$  can be defined as the least partial ordering over  $Term_{\perp, \text{F}}$  satisfying the following properties:

- $\perp \sqsubseteq e$  for all  $e \in Term_{\perp, \text{F}}$ ,
- $h(e_1, \dots, e_n) \sqsubseteq h(e'_1, \dots, e'_n)$ , if  $e_i \sqsubseteq e'_i$  for all  $i \in \{1, \dots, n\}$ ,  $h \in DC \cup FS \cup \{\text{fails}\} \cup \{\text{F}\}$

The intended meaning of  $e \sqsubseteq e'$  is that  $e$  is less defined or has less information than  $e'$ . Two expressions  $e, e' \in Term_{\perp, \text{F}}$  are *consistent* if they can be refined to obtain the same information, i.e., if there exists  $e'' \in Term_{\perp, \text{F}}$  such that  $e \sqsubseteq e''$  and  $e' \sqsubseteq e''$ . Notice that according to this  $\text{F}$  is maximal. This is reasonable from an intuitive point of view, since  $\text{F}$  represents ‘failure of reduction’ for an expression and this gives a no further refinable information about the result of the evaluation of such expression. This contrasts with the status given to failure in [16], where  $\text{F}$  is chosen to verify  $\text{F} \sqsubseteq t$  for any  $t$  different from  $\perp$ .

We also consider the relation  $\sqsubseteq$  referred to substitutions:  $\sigma \sqsubseteq \sigma'$  iff  $X\sigma \sqsubseteq X\sigma'$  for all  $X \in \mathcal{V}$ . And for tuples of cterms:  $\vec{t} \sqsubseteq \vec{t}'$  iff the ordering relation is pairwise satisfied.

Extending  $\sqsubseteq$  to sets of terms results in the *Egli-Milner* preordering (see e.g. [17]): given  $D, D' \subseteq CTerm_{\perp, \text{F}}$  we say that  $D'$  is more refined than  $D$  and write  $D \sqsubseteq D'$  iff for all  $t \in D$  there exists  $t' \in D'$  with  $t \sqsubseteq t'$  and for all  $t' \in D'$  there exists  $t \in D$  with  $t \sqsubseteq t'$ . The sets  $D$  and  $D'$  are **consistent** iff there exists  $D''$  such that  $D \sqsubseteq D''$  and  $D' \sqsubseteq D''$ .

### 3 A Semantic Framework for *FLP* with Failure

We present here (a slight variant of) the set-oriented approach to *FLP* and failure of [13].

### 3.1 Set expressions

A set-expression is a syntactical construction designed for manipulating sets of values. A (total) set-expression  $\mathcal{S}$  is defined as:

$$\mathcal{S} ::= \{t\} \mid f(\bar{t}) \mid \text{fails}(\mathcal{S}_1) \mid \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2 \mid \mathcal{S}_1 \cup \mathcal{S}_2$$

where  $t \in CTerm_{\mathbb{F}}$ ,  $\bar{t} \in CTerm_{\mathbb{F}} \times \dots \times CTerm_{\mathbb{F}}$ ,  $f \in FS^n$ , and  $\mathcal{S}_1, \mathcal{S}_2$  are set-expressions. We write  $SetExp$  for the set of (total) set-expressions. The set  $SetExp_{\perp}$  of *partial* set-expressions have the same syntax except that the cterms  $t, \bar{t}$  can contain  $\perp$ .

Indexed unions bind variables in a similar way to other more familiar constructs like first order quantifications or  $\lambda$ -abstraction. The set  $PV(\mathcal{S})$  of bound or *produced variables* of a set-expression  $\mathcal{S}$  can be formally defined as:

$$\begin{aligned} \bullet PV(\{t\}) &= PV(f(\bar{t})) = \emptyset & \bullet PV(\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2) &= \{X\} \cup PV(\mathcal{S}_1) \cup PV(\mathcal{S}_2) \\ \bullet PV(\text{fails}(\mathcal{S})) &= PV(\mathcal{S}) & \bullet PV(\mathcal{S}_1 \cup \mathcal{S}_2) &= PV(\mathcal{S}_1) \cup PV(\mathcal{S}_2) \end{aligned}$$

We can also define the set  $FV(\mathcal{S})$  of *free variables* of a set-expression  $\mathcal{S}$  as:

$$\begin{aligned} \bullet FV(\{t\}) &= \text{var}(t) & \bullet FV(f(\bar{t})) &= \text{var}(\bar{t}) & \bullet FV(\text{fails}(\mathcal{S}_1)) &= FV(\mathcal{S}_1) \\ \bullet FV(\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2) &= (FV(\mathcal{S}_2) - \{X\}) \cup FV(\mathcal{S}_1) \\ \bullet FV(\mathcal{S}_1 \cup \mathcal{S}_2) &= FV(\mathcal{S}_1) \cup FV(\mathcal{S}_2) \end{aligned}$$

In order to avoid variable renamings and simplify further definitions, we add an *admissibility condition* to set expressions: for  $\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2$  we require  $X \notin \text{var}(\mathcal{S}_1) \cup PV(\mathcal{S}_2)$  and  $PV(\mathcal{S}_1) \cap PV(\mathcal{S}_2) = \emptyset$ ; and  $\mathcal{S}_1 \cup \mathcal{S}_2$  must verify  $PV(\mathcal{S}_1) \cap FV(\mathcal{S}_2) = \emptyset$  and  $PV(\mathcal{S}_2) \cap FV(\mathcal{S}_1) = \emptyset$ . Notice that with this conditions, the sets  $PV(\mathcal{S})$  and  $FV(\mathcal{S})$  define a partition over  $\text{var}(\mathcal{S})$ . In the following we always assume this admissibility condition over set-expressions.

As an example, consider a function  $f \in FS^2$  and a constructor  $c \in DC^2$ . The next is an admissible set-expression:

$$\mathcal{S} = \bigcup_{A \in \bigcup_{B \in f(X,Y)} \{B\}} \{c(A, X)\} \cup \bigcup_{C \in \{X\}} f(C, Y)$$

It is easy to check that  $PV(\mathcal{S}) = \{A, B, C\}$  and  $FV(\mathcal{S}) = \{X, Y\}$

We assume also some admissibility conditions over substitutions: given a set-expression  $\mathcal{S}$  we say that  $\sigma$  is an *admissible substitution for  $\mathcal{S}$* , if  $Dom(\sigma) \cap PV(\mathcal{S}) = \emptyset$  and  $Ran(\sigma) \cap PV(\mathcal{S}) = \emptyset$ . In such case, the set-expression  $\mathcal{S}\sigma$  is naturally defined as:

$$\begin{aligned} \bullet \{t\}\sigma &= \{t\sigma\} & \bullet f(\bar{t})\sigma &= f(\bar{t}\sigma) & \bullet \text{fails}(\mathcal{S})\sigma &= \text{fails}(\mathcal{S}\sigma) \\ \bullet (\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2)\sigma &= \bigcup_{X \in \mathcal{S}_1\sigma} \mathcal{S}_2\sigma & \bullet (\mathcal{S}_1 \cup \mathcal{S}_2)\sigma &= \mathcal{S}_1\sigma \cup \mathcal{S}_2\sigma \end{aligned}$$

Notice that produced variables are not affected by the substitution due to the condition  $Dom(\sigma) \cap PV(\mathcal{S}) = \emptyset$ ; and we avoid capture of produced variables with the condition  $Ran(\sigma) \cap PV(\mathcal{S}) = \emptyset$ . We will also use (admissible) *set-substitutions* for set-expressions: given a set  $D = \{s_1, \dots, s_n\} \subseteq CTerm_{\perp, \mathbb{F}}$  we write  $\mathcal{S}[Y/D]$  as a shorthand for the distribution  $\mathcal{S}[Y/s_1] \cup \dots \cup \mathcal{S}[Y/s_n]$ . Extending this notation, we also write  $\mathcal{S}[X_1/D_1, \dots, X_n/D_n]$  (where  $D_1, \dots, D_n \in CTerm_{\perp, \mathbb{F}}$ ) as a shorthand for  $(\dots(\mathcal{S}[X_1/D_1])\dots)[X_n/D_n]$ . In the following all the substitutions we use are admissible.

The ordering  $\sqsubseteq$  defined in Section 2 for terms and sets of terms can be extended also to set expressions: the relation  $\sqsubseteq_{SetExp_{\perp}}$  is the least reflexive and transitive relation satisfying:

1.  $\{\perp\} \sqsubseteq_{SetExp_{\perp}} \mathcal{S}$ , for any  $\mathcal{S} \in SetExp_{\perp}$
2.  $\{t\} \sqsubseteq_{SetExp_{\perp}} \{t'\}$ , if  $t \sqsubseteq t'$
3.  $f(\bar{t}) \sqsubseteq_{SetExp_{\perp}} f(\bar{t}')$ , if  $f(\bar{t}) \sqsubseteq f(\bar{t}')$
4.  $fails(\mathcal{S}) \sqsubseteq_{SetExp_{\perp}} fails(\mathcal{S}')$ , if  $\mathcal{S}' \sqsubseteq_{SetExp_{\perp}} \mathcal{S}'$
5.  $\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2 \sqsubseteq_{SetExp_{\perp}} \bigcup_{X \in \mathcal{S}'_1} \mathcal{S}'_2$ , if  $\mathcal{S}_1 \sqsubseteq_{SetExp_{\perp}} \mathcal{S}'_1$  and  $\mathcal{S}_2 \sqsubseteq_{SetExp_{\perp}} \mathcal{S}'_2$
6.  $\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n \sqsubseteq_{SetExp_{\perp}} \mathcal{S}'_1 \cup \dots \cup \mathcal{S}'_m$ , if for all  $i \in \{1, \dots, n\}$  there exists some  $j \in \{1, \dots, m\}$  such that  $\mathcal{S}_i \sqsubseteq \mathcal{S}_j$  and conversely, for all  $j \in \{1, \dots, m\}$  there exists some  $i \in \{1, \dots, n\}$  such that  $\mathcal{S}_i \sqsubseteq \mathcal{S}_j$

In the following the subindex of  $\sqsubseteq_{SetExp_{\perp}}$  will be omitted.

Any term  $e \in Term_{\perp, \mathbb{F}}$  has a corresponding set-expression  $\widehat{e} \in SetExp_{\perp, \mathbb{F}}$ , that can be defined as:

- $\widehat{X} = \{X\}$
- $\widehat{fails(e)} = fails(\widehat{e})$
- $\widehat{c(e_1, \dots, e_n)} = \bigcup_{X_1 \in \widehat{e}_1} \dots \bigcup_{X_n \in \widehat{e}_n} \{c(X_1, \dots, X_n)\}$ , for every  $c \in DC^n \cup \{\perp, \mathbb{F}\}$ , where the  $X_1, \dots, X_n$  are fresh variables
- $\widehat{f(e_1, \dots, e_n)} = \bigcup_{X_1 \in \widehat{e}_1} \dots \bigcup_{X_n \in \widehat{e}_n} f(X_1, \dots, X_n)$ , for every  $f \in FS^n$ , where  $X_1, \dots, X_n$  are fresh variables

### 3.2 Programs

A program is a set of rules of the form:

$$f(t_1, \dots, t_n) \rightarrow \mathcal{S}$$

where  $f \in FS^n$ ,  $t_i \in CTerm$ , the tuple  $(t_1, \dots, t_n)$  is linear (each variable occurs only once),  $\mathcal{S} \in SetExp$  and  $FV(\mathcal{S}) \subseteq var((t_1, \dots, t_n))$ .

Notice that we allow  $\mathbb{F}$  to appear in  $\mathcal{S}$ , but not in  $t_1, \dots, t_n$ . This is not essential, and is done only for technical convenience.

Notice also that known definitions which refer only to heads of rules, like that of *definitional tree* or *inductively sequential program* [2], can be applied also to our programs. Concretely, we are interested in the class of *Complete Inductively Sequential Programs* (*CIS*-programs for short), introduced in [2]. ‘Complete’ means that at every *case* distinction in the definitional tree of a function there is a branch for every constructor symbol from *DC*. By considering *CIS*-programs we ensure that, for any ground  $t_1, \dots, t_n \in CTerm$ , exactly *one* program rule can be used to reduce a call  $f(t_1, \dots, t_n)$ . And there is no loss of generality: in [3, 13] it is shown how to convert programs into *overlapping* inductively sequential programs, where several rules might have the same head; as mentioned in [13], by using  $\cup$  it is straightforward to achieve inductive sequentiality, just by merging with  $\cup$  several body rules; in [13] it is also shown how to translate ‘classical’ syntax for expressions in bodies into set-oriented syntax; finally, to achieve completeness we only need to add, for every ‘missing’ constructor in a *case* distinction of a definitional tree, a rule with  $\{\mathbb{F}\}$  as body.

As an example, the program of Sect. 1 admits the following writing as a *CIS*-program:

$$\begin{aligned}
& \text{next}(a) \rightarrow \{b, c\} & \text{next}(c) \rightarrow \{\mathbb{F}\} \\
& \text{next}(b) \rightarrow \{c, d\} & \text{next}(d) \rightarrow \{\mathbb{F}\} \\
& \text{path}(X, Y) \rightarrow \bigcup_{A \in \text{eq}(X, Y)} \bigcup_{B \in \bigcup_{C \in \text{next}(X)} \text{path}(C, Y)} \text{ifThenElse}(A, \text{true}, B) \\
& \text{safe}(X) \rightarrow \text{fails}\left(\bigcup_{A \in \text{eq}(X, d)} \bigcup_{B \in \bigcup_{C \in \text{next}(X)} \text{path}(C, d)} \text{ifThenElse}(A, \text{true}, B)\right)
\end{aligned}$$

In practice, of course, programmers could be alleviated of such cumbersome syntax by automating the translation.

### 3.3 Proof calculus for programs and set-expressions

Table 1 shows the *Set Reduction Logic* that defines the semantics of set-expressions with respect to *CIS*-programs.

**Table 1.** Rules for *SRL*-provability

(1) $\frac{}{\mathcal{S} \triangleleft \{\perp\}} \quad \mathcal{S} \in \text{SetExp}_{\perp}$	(2) $\frac{}{\{X\} \triangleleft \{X\}} \quad X \in \mathcal{V}$
(3) $\frac{\{t_1\} \triangleleft \mathcal{C}_1 \quad \{t_n\} \triangleleft \mathcal{C}_n}{\{c(t_1, \dots, t_n)\} \triangleleft \{c(\vec{t}) \mid \vec{t} \in \mathcal{C}_1 \times \dots \times \mathcal{C}_n\}} \quad c \in DC \cup \{\mathbb{F}\}$	
(4) $\frac{\mathcal{S} \triangleleft \mathcal{C}}{f(\vec{t}) \triangleleft \mathcal{C}} \quad (f(\vec{t}) \rightarrow \mathcal{S}) \in [\widehat{\mathcal{P}}]_{\perp, \mathbb{F}}$	
(5) $\frac{}{f(\vec{t}) \triangleleft \{\mathbb{F}\}} \quad \text{for all } (f(\vec{s}) \rightarrow \mathcal{S}') \in \widehat{\mathcal{P}}, \vec{t} \text{ and } \vec{s} \text{ have a } DC \cup \{\mathbb{F}\}\text{-clash}$	
(6) $\frac{\mathcal{S} \triangleleft \{\mathbb{F}\}}{\text{fails}(\mathcal{S}) \triangleleft \{\text{true}\}}$	
(7) $\frac{\mathcal{S} \triangleleft \mathcal{C}}{\text{fails}(\mathcal{S}) \triangleleft \{\text{false}\}} \quad \text{there is some } t \in \mathcal{C}, t \neq \perp, t \neq \mathbb{F}$	
(8) $\frac{\mathcal{S}_1 \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2[X/\mathcal{C}_1] \triangleleft \mathcal{C}}{\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2 \triangleleft \mathcal{C}}$	(9) $\frac{\mathcal{S}_1 \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2 \triangleleft \mathcal{C}_2}{\mathcal{S}_1 \cup \mathcal{S}_2 \triangleleft \mathcal{C}_1 \cup \mathcal{C}_2}$

Rules 1 to 4 are “classical” in *CRWL(F)* [8, 13, 12]. Rule 4 uses a *c*-instance of a program rule; the set of such *c*-instances is defined as:  $[\widehat{\mathcal{P}}]_{\perp, \mathbb{F}} = \{R\theta \mid R = (f(\vec{t}) \rightarrow \mathcal{S}) \in \widehat{\mathcal{P}}, \theta \in C\text{Subst}_{\perp, \mathbb{F}}\}$ . Notice that this *c*-instance is unique if it exists (due to the non-overlapping condition imposed to programs). If such *c*-instance does not exist, then by rule 5, the corresponding set-expression reduces to  $\{\mathbb{F}\}$ .

Rules 6 and 7 establish the meaning of the function  $\text{fails}(\mathcal{S})$ : we must reduce  $\mathcal{S}$ ; if we achieve  $\{\mathbb{F}\}$  as a *SAS* for it, this means that any attempt to reduce  $\mathcal{S}$  effectively fails. On the other hand, if we obtain a *SAS* with a *c*term of the form  $c(\dots)$  or  $X$ , then there is a possible reduction of  $\mathcal{S}$  to a *c*term. This is a “constructive” way of proving failure. Moreover, the unique *SAS*’s for  $\mathcal{S}$  that do not provide enough information for reducing  $\text{fails}(\mathcal{S})$  are  $\{\perp\}$  or  $\{\perp, \mathbb{F}\}$ . Finally, rules 8 and 9 are natural to understand from classical set theory.

Given a program  $\mathcal{P}$  and  $\mathcal{S} \in \text{SetExp}$  we write  $\mathcal{P} \vdash_{SRL} \mathcal{S} \triangleleft \mathcal{C}$  if the relation  $\mathcal{S} \triangleleft \mathcal{C}$  is provable with respect to *CRWL* and the program  $\mathcal{P}$ . The denotation of  $\mathcal{S}$  is defined as  $\llbracket \mathcal{S} \rrbracket = \{\mathcal{C} \mid \mathcal{S} \triangleleft \mathcal{C}\}$ . Then the denotation of a set-expression is a set of sets of (possible partial) cterms.

It is easy to check that the symbol  $\cup$  is associative and commutative, i.e.,  $\llbracket (\mathcal{S}_1 \cup \mathcal{S}_2) \cup \mathcal{S}_3 \rrbracket = \llbracket \mathcal{S}_1 \cup (\mathcal{S}_2 \cup \mathcal{S}_3) \rrbracket$  and  $\llbracket \mathcal{S}_1 \cup \mathcal{S}_2 \rrbracket = \llbracket \mathcal{S}_2 \cup \mathcal{S}_1 \rrbracket$ . So, in the following we avoid unnecessary parentheses and consider the expression  $\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n$  as equivalent to  $\mathcal{S}_{i_1} \cup \dots \cup \mathcal{S}_{i_n}$ , where  $(i_1, \dots, i_n)$  is any permutation of  $(1, \dots, n)$ . Moreover,  $\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n \triangleleft \mathcal{C}$  iff  $\mathcal{S}_1 \triangleleft \mathcal{C}_1, \dots, \mathcal{S}_n \triangleleft \mathcal{C}_n$ , where  $\mathcal{C} = \mathcal{C}_1 \cup \dots \cup \mathcal{C}_n$ .

In some results we will consider an equivalent formulation for rule 9:

$$(9') \quad \frac{\mathcal{S}_1 \triangleleft \mathcal{C}_1 \quad \dots \quad \mathcal{S}_n \triangleleft \mathcal{C}_n}{\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n \triangleleft \mathcal{C}_1 \cup \dots \cup \mathcal{C}_n}$$

of course, if  $n > 1$ .

We have said that produced variables in a set-expression plays the role of bounded variables. Then such variables can not appear in any *SAS* of the set expression. Formally:

**Proposition 1.** *If  $\mathcal{S} \triangleleft \mathcal{C}$  then  $\text{var}(\mathcal{C}) \cap PV(\mathcal{S}) = \emptyset$ .*

*Proof.* It is not difficult to formalize this proof by i.h. on the length of the proof (number of rules applied) for  $\mathcal{S} \triangleleft \mathcal{C}$  taking into account the admissibility conditions on set-expressions.

The denotation of any cterm has an simple characterization as shows the next property:

**Proposition 2.** *For every  $t \in CTerm_{\perp, F}$  we have  $\llbracket \{t\} \rrbracket = \{\{s\} \mid s \in CTerm_{\perp, F}, s \sqsubseteq t\}$*

*Proof.* By induction on the structure of  $t$ :

- if  $t = \perp$  the result is trivial;
- if  $t = \{X\}$  then  $\llbracket \{t\} \rrbracket = \{\{\perp\}, \{X\}\}$  and the result is clear;
- if  $t = c \in DC^0 \cup \{F\}$  then  $\llbracket \{t\} \rrbracket = \{\{\perp\}, \{c\}\}$  and the result is clear;
- if  $t = c(t_1, \dots, t_n)$  then by i.h. (the structure of each  $t_i$  is simpler than  $t$ ) we have  $\llbracket \{t_i\} \rrbracket = \{s_i \mid s_i \in CTerm_{\perp, F}, s_i \sqsubseteq t_i\}$ . Every *SAS* in the denotation of  $t$  is built by rules 1 and 3 of *CRWL*, and we have  $\llbracket \{c(t_1, \dots, t_n)\} \rrbracket = \{\perp\} \cup \{c(t'_1, \dots, t'_n) \mid t'_i \in \llbracket \{t_i\} \rrbracket\}$ , what coincides with the set of cterms  $s$  such that  $s \sqsubseteq t$ .

In order to prove the results of Section 4 we have needed to generalize some results of [13] and to prove new ones. In particular, Theorem 1 below is a key technical result about the *SRL* calculus.

**Lemma 1 (Lifting).** *Let  $\sigma, \sigma' \in CSubst_{\perp, F}$  with  $\sigma \sqsubseteq \sigma'$ . If  $\mathcal{S}\sigma \triangleleft \mathcal{C}$  then there exist a proof in *CRWL* for  $\mathcal{S}\sigma' \triangleleft \mathcal{C}$  with the same length and structure.*

*Proof.* We proceed by induction on the length of the proof for  $\mathcal{S}\sigma \triangleleft \mathcal{C}$ :  
 $\underline{l=0}$  The proof can be:

- $\mathcal{S} \triangleleft \{\perp\}$  by rule 1: trivial.
- $\{X\} \triangleleft \{X\}$  by rule 2; if  $\mathcal{S}\sigma = \{X\}$  it must be  $\mathcal{S}\sigma' = \{X\}$  and the result is direct.
- $\{c\} \triangleleft \{c\}$  with  $\mathcal{S}\sigma = \{c\}$  and  $c \in DC^0 \cup \{\mathbb{F}\}$ ; again it must be  $\mathcal{S}\sigma' = \{c\}$  and the result is clear.
- $f(\vec{t}) \triangleleft \{\mathbb{F}\}$  by rule 5. As  $\sigma$  cannot introduce any function symbol it must be  $\mathcal{S} = f(\vec{s})$ . Then  $f(\vec{s})\sigma \sqsubseteq f(\vec{t})\sigma'$  and it is clear that if  $f(\vec{s})\sigma$  clash with any rule of the program then also clash  $f(\vec{s})\sigma$ . Then we have  $f(\vec{s})\sigma' \triangleleft \{\mathbb{F}\}$  by the same rule 5.

$l+1$  The proof can be:

- if the proof is done by rule 3 in  $l+1$  steps then  $\mathcal{S}\sigma = \{c(t_1, \dots, t_n)\}$  and the proof will have the form:

$$\frac{\{t_1\} \triangleleft \mathcal{C}_1 \quad \{t_n\} \triangleleft \mathcal{C}_n}{\{c(t_1, \dots, t_n)\} \triangleleft \{c(\vec{t}) \mid \vec{t}' \in \mathcal{C}_1 \times \dots \times \mathcal{C}_n\}}$$

It must be  $\mathcal{S}\sigma' = \{c(t'_1, \dots, t'_n)\}$  with  $c(t_1, \dots, t_n) \sqsubseteq c(t'_1, \dots, t'_n)$ . By i.h. for each  $i$  we have a proof  $\{t'_i\} \triangleleft \mathcal{C}_i$  with the same length and structure. Then by rule 3 we have:

$$\frac{\{t'_1\} \triangleleft \mathcal{C}_1 \quad \{t'_n\} \triangleleft \mathcal{C}_n}{\{c(t'_1, \dots, t'_n)\} \triangleleft \{c(\vec{t}') \mid \vec{t}' \in \mathcal{C}_1 \times \dots \times \mathcal{C}_n\}}$$

- if the proof is done by rule 4, then it must be  $\mathcal{S}\sigma = f(\vec{t})$  and  $\mathcal{S}\sigma' = f(\vec{t}')$  with  $f(\vec{t}) \sqsubseteq f(\vec{t}')$ . For applying rule 4 we must have a rule  $f(\vec{s}') \rightarrow \mathcal{S}'$  in  $\mathcal{P}$  and a substitution  $\theta$  such that  $f(\vec{s}')\theta = f(\vec{t})$ . It is easy to see that then there exists  $\theta' \sqsubseteq \theta$  such that  $f(\vec{s}')\theta' = f(\vec{t}')$  and then the same rule 4 with a more refined c-instance is applicable. The  $\mathcal{S}\mathcal{A}\mathcal{S}$ 's for both expression are obtained from the c-instances of the body  $\mathcal{S}'\theta$  and  $\mathcal{S}'\theta'$  respectively. As  $\theta' \sqsubseteq \theta$  then we are in the hypothesis of induction that ensures that if  $\mathcal{S}'\theta \triangleleft \mathcal{C}$  then we have a proof for  $\mathcal{S}'\theta' \triangleleft \mathcal{C}$  with the same length and structure. And then we have proofs for  $f(\vec{t}) \triangleleft \mathcal{C}$  and  $f(\vec{t}') \triangleleft \mathcal{C}$  with the same length and structure.
- if the proof is done by rule 6 in  $l+1$  steps, then then it must be  $\mathcal{S} = \text{fails}(\mathcal{S}')$  and  $\mathcal{S}\sigma = \text{fails}(\mathcal{S}'\sigma)$ , with  $\mathcal{S}'\sigma \triangleleft \{\mathbb{F}\}$ . By i.h. we also have a proof for  $\mathcal{S}'\sigma' \triangleleft \{\mathbb{F}\}$  with the same length and structure, and then clearly we have proofs for  $\mathcal{S}\sigma = \{\{\perp\}, \{\mathbb{F}\}\}$  and  $\mathcal{S}\sigma' = \{\{\perp\}, \{\mathbb{F}\}\}$  with the same length and structure.
- if the proof is done by rule 7 in  $l+1$  steps, again we have  $\mathcal{S} = \text{fails}(\mathcal{S}')$ ,  $\mathcal{S}\sigma = \text{fails}(\mathcal{S}'\sigma)$  and  $\mathcal{S}\sigma' = \text{fails}(\mathcal{S}'\sigma')$ . If  $\text{fails}(\mathcal{S}'\sigma) \triangleleft \{\text{true}\}$  then it must be  $\mathcal{S}'\sigma \triangleleft \mathcal{C}$  with  $t \in \mathcal{C}$ ,  $t \neq \perp$  and  $t \neq \mathbb{F}$ . By i.h. there exist a proof for  $\mathcal{S}'\sigma' \triangleleft \mathcal{C}$  with the same length and structure and then there also exist a proof for  $\text{fails}(\mathcal{S}'\sigma') \triangleleft \{\text{true}\}$  with the same length and structure of  $\text{fails}(\mathcal{S}'\sigma) \triangleleft \{\text{true}\}$ .
- if the proof is done by rule 8, then it must be  $\mathcal{S} = \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2$  and the proof for  $\mathcal{S}\sigma \triangleleft \mathcal{C}$  must be:

$$\frac{\mathcal{S}_1\sigma \triangleleft \mathcal{C}' \quad \mathcal{S}_2\sigma[X/\mathcal{C}'] \triangleleft \mathcal{C}}{\bigcup_{X \in \mathcal{S}_1\sigma} \mathcal{S}_2\sigma \triangleleft \mathcal{C}}$$

By i.h. we have a proof for  $\mathcal{S}_1\sigma' \triangleleft \mathcal{C}'$  with the same length and structure of  $\mathcal{S}_1\sigma \triangleleft \mathcal{C}'$ . If  $\mathcal{C}' = \{t_1, \dots, t_n\}$  the notation  $\mathcal{S}_2\sigma[X/\mathcal{C}']$  is a short for  $\mathcal{S}_2\sigma[X/t_1] \cup \dots \cup \mathcal{S}_2\sigma[X/t_n]$  and  $\mathcal{S}_2\sigma'[X/\mathcal{C}']$  is a short for  $\mathcal{S}_2\sigma'[X/t_1] \cup \dots \cup \mathcal{S}_2\sigma'[X/t_n]$ . It is not difficult to see that  $\sigma[X/t_i] \sqsubseteq \sigma'[X/t_i]$ , and then these substitutions satisfy the i.h., i.e., if  $\mathcal{S}_2\sigma[X/t_1] \cup \dots \cup \mathcal{S}_2\sigma[X/t_n] \triangleleft \mathcal{C}$  then we also have proofs for  $\mathcal{S}_2\sigma'[X/t_1] \cup \dots \cup \mathcal{S}_2\sigma'[X/t_n] \triangleleft \mathcal{C}$  with the same length and structure, and then  $\bigcup_{X \in \mathcal{S}_1\sigma'} \mathcal{S}_2\sigma' \triangleleft \mathcal{C}$ .

- if the proof is done by rule 9 then it must be  $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$  and the proof for  $\mathcal{S}\sigma \triangleleft \mathcal{C}$  must be:

$$\frac{\mathcal{S}_1\sigma \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2\sigma \triangleleft \mathcal{C}_2}{\mathcal{S}_1\sigma \cup \mathcal{S}_2\sigma \triangleleft \mathcal{C}_1 \cup \mathcal{C}_2}$$

By i.h. we have proofs for  $\mathcal{S}_1\sigma' \triangleleft \mathcal{C}_1$  and  $\mathcal{S}_2\sigma' \triangleleft \mathcal{C}_2$  with the same length and structure and then, by rule 9 we have  $\mathcal{S}_1\sigma' \cup \mathcal{S}_2\sigma' \triangleleft \mathcal{C}_1 \cup \mathcal{C}_2$ .  $\square$

An easy consequence of this result is:

**Corollary 1.** *If  $t \sqsubseteq t'$  and  $\mathcal{S}[X/t] \triangleleft \mathcal{C}$  then  $\mathcal{S}[X/t'] \triangleleft \mathcal{C}$ .*

*Proof.* Trivial from Lemma 1 taking  $\sigma = [X/t]$  and  $\sigma' = [X/t']$ .

As a consequence of this Corollary 1 one could expect to get the extended following version: if  $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$  and  $\mathcal{S}[X/\mathcal{C}_1] \triangleleft \mathcal{C}$ , then  $\mathcal{S}[X/\mathcal{C}_2] \triangleleft \mathcal{C}$ . But this is not true. For example let  $\mathcal{S} = X$  and  $\mathcal{C}_1 = \{c(0, \perp), c(\perp, 1)\}$ ,  $\mathcal{C}_2 = \{c(0, 1)\}$  ( $c \in DC^2$ ). It is easy to see  $\mathcal{S}[X/\mathcal{C}_1] \triangleleft \mathcal{C}_1$  and  $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$ , but it is not true that  $\mathcal{S}[X/\mathcal{C}_2] \triangleleft \mathcal{C}_1$ . So, the extension is not direct, but it is still possible as we will see in Th. 1.

The following results establish some properties of the relation  $\sqsubseteq$  for set-expressions and they will be useful for proving Th. 1.

First of all an easy observation: a general set-expression can always be decomposed into an equivalent  $\cup$ -expression of the form  $\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n$  ( $n \geq 1$ ) where  $\mathcal{S}_1, \dots, \mathcal{S}_n$  are **single set-expressions** of the form  $\{t\}$  or  $f(\bar{t})$  or  $fails(\dots)$  or  $\bigcup_{X \in \dots} \dots$ , i.e.,  $\mathcal{S}_1, \dots, \mathcal{S}_n$  have not any  $\cup$  at the external structure. In the following we will use such decomposition into single set-expressions several times.

**Proposition 3.** *If  $\mathcal{S} \sqsubseteq \mathcal{S}'$  and  $t \sqsubseteq t'$  then  $\mathcal{S}[X/t] \sqsubseteq \mathcal{S}'[X/t']$*

*Proof.* In general we can consider  $\mathcal{S} = \mathcal{S}_1 \cup \dots \cup \mathcal{S}_n$  and  $\mathcal{S}' = \mathcal{S}'_1 \cup \dots \cup \mathcal{S}'_m$ , where each  $\mathcal{S}_i$  and each  $\mathcal{S}_j$  is a single set-expression, such that for any  $i \in \{1, \dots, n\}$  there exists some  $j \in \{1, \dots, m\}$  such that  $\mathcal{S}_i \sqsubseteq \mathcal{S}_j$  and conversely. It is enough to prove that if  $\mathcal{S}_i \sqsubseteq \mathcal{S}'_j$  then  $\mathcal{S}_i[X/t] \sqsubseteq \mathcal{S}'_j[X/t']$ . In general, if  $\mathcal{S}$  and  $\mathcal{S}'$  are single set-expressions such that  $\mathcal{S} \sqsubseteq \mathcal{S}'$  and  $t \sqsubseteq t'$ , then  $\mathcal{S}[X/t] \sqsubseteq \mathcal{S}'[X/t']$

- if  $\mathcal{S} = \{s\}$  then it must be  $\mathcal{S}' = \{s'\}$  with  $s \sqsubseteq s'$ , and it is easy to see that  $\{s[X/t]\} \sqsubseteq \{s[X/t']\}$
- if  $\mathcal{S} = f(\bar{s})$  then  $\mathcal{S}' = f(\bar{s}')$  with  $\bar{s} \sqsubseteq \bar{s}'$ . As before it is easy to prove  $\bar{s}[X/t] \sqsubseteq \bar{s}'[X/t']$
- if  $\mathcal{S} = fails(\mathcal{S}_1)$  then  $\mathcal{S}' = fails(\mathcal{S}'_1)$  with  $\mathcal{S}_1 \sqsubseteq \mathcal{S}'_1$ . By i.h.  $\mathcal{S}_1[X/t] \sqsubseteq \mathcal{S}'_1[X/t']$  and then  $fails(\mathcal{S}_1)[X/t] \sqsubseteq fails(\mathcal{S}'_1)[X/t']$
- if  $\mathcal{S} = \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2$ , then  $\mathcal{S}' = \bigcup_{X \in \mathcal{S}'_1} \mathcal{S}'_2$ , with  $\mathcal{S}_1 \sqsubseteq \mathcal{S}'_1$  and  $\mathcal{S}_2 \sqsubseteq \mathcal{S}'_2$ . By i.h.  $\mathcal{S}_1[X/t] \sqsubseteq \mathcal{S}'_1[X/t']$  and  $\mathcal{S}_2[X/t] \sqsubseteq \mathcal{S}'_2[X/t']$ . As  $\mathcal{S}[X/t] = \bigcup_{X \in \mathcal{S}_1[X/t]} \mathcal{S}_2[X/t]$  and  $\mathcal{S}'[X/t'] = \bigcup_{X \in \mathcal{S}'_1[X/t']} \mathcal{S}'_2[X/t']$ , the result is direct.

**Corollary 2.** *If  $\mathcal{S} \sqsubseteq \mathcal{S}'$  and  $\sigma \sqsubseteq \sigma'$  ( $\sigma, \sigma' \in CSubst_{\perp, F}$ ) then  $\mathcal{S}\sigma \sqsubseteq \mathcal{S}'\sigma'$*

*Proof.* The substitutions can be expressed as  $\sigma = [X_1/t_1, \dots, X_n/t_n]$  and  $\sigma' = [X_1/t'_1, \dots, X_n/t'_n]$ , where  $\bar{t} \sqsubseteq \bar{t}'$ . Then, this result comes from successive applications of Prop. 3 to pairs of substitutions  $[X_1/t_1]$  and  $[X_1/t'_1], \dots, [X_1/t_n]$  and  $[X_1/t'_n]$

**Proposition 4.** *If  $\mathcal{S} \sqsubseteq \mathcal{S}'$  and  $\mathcal{C} \sqsubseteq \mathcal{C}'$  then  $\mathcal{S}[X/\mathcal{C}] \sqsubseteq \mathcal{S}'[X/\mathcal{C}']$*

*Proof.* Assume  $\mathcal{C} = \{t_1, \dots, t_n\}$  and  $\mathcal{C}' = \{s_1, \dots, s_m\}$ . Similar to the previous property, we consider  $\mathcal{S} = \mathcal{S}_1 \cup \dots \cup \mathcal{S}_p$  and  $\mathcal{S}' = \mathcal{S}'_1 \cup \dots \cup \mathcal{S}'_q$ , where each  $\mathcal{S}_i$  and each  $\mathcal{S}_j$  are single set-expressions, and for any  $i \in \{1, \dots, n\}$  there exists some  $j \in \{1, \dots, m\}$  such that  $\mathcal{S}_i \sqsubseteq \mathcal{S}_j$  and conversely. The set substitutions produce:

$$\begin{aligned} \mathcal{S}[X/\mathcal{C}] &= \mathcal{S}_1[X/t_1] \cup \dots \cup \mathcal{S}_1[X/t_n] \cup \dots \cup \mathcal{S}_p[X/t_1] \cup \dots \cup \mathcal{S}_p[X/t_n] \\ \mathcal{S}'[X/\mathcal{C}'] &= \mathcal{S}'_1[X/s_1] \cup \dots \cup \mathcal{S}'_1[X/s_m] \cup \dots \cup \mathcal{S}'_q[X/s_1] \cup \dots \cup \mathcal{S}'_q[X/s_m] \end{aligned}$$

From Prop. 3  $\mathcal{S}_i \sqsubseteq \mathcal{S}_j$  and  $t_k \sqsubseteq s_l$ , then  $\mathcal{S}_i[X/t_k] \sqsubseteq \mathcal{S}'_j[X/s_l]$ . Then it is easy to conclude that  $\mathcal{S}[X/\mathcal{C}] \sqsubseteq \mathcal{S}'[X/\mathcal{C}']$ .

The next Lemma is basically the same as Th. 1 except that here we restrict to single set-expressions.

**Lemma 2 (Refinement).** *Let  $\mathcal{S}_1, \dots, \mathcal{S}_n, \mathcal{S}$  be single set-expressions such that  $\mathcal{S}_1 \sqsubseteq \mathcal{S}, \dots, \mathcal{S}_n \sqsubseteq \mathcal{S}$ . Then*

$$\mathcal{S}_1 \triangleleft \mathcal{C}_1, \dots, \mathcal{S}_n \triangleleft \mathcal{C}_n \Rightarrow \mathcal{S} \triangleleft \mathcal{C}$$

for some  $\mathcal{C}$  such that  $\mathcal{C}_1 \sqsubseteq \mathcal{C}, \dots, \mathcal{C}_n \sqsubseteq \mathcal{C}$ .

*Proof.* First of all, notice that  $\mathcal{S}_1, \dots, \mathcal{S}_n$  are consistent set-expressions, in fact refined by  $\mathcal{S}$ . In general, for some of the set-expressions we can have  $\mathcal{S}_i = \{\perp\}$  or, even although  $\mathcal{S}_i \neq \{\perp\}$ , the proof  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  can be trivially done by rule 1 producing also  $\mathcal{C}_i = \{\perp\}$ . This *SAS* is refined by any other, in particular by any  $\mathcal{C}$  obtained for  $\mathcal{S}$ . The interesting cases the proofs  $\mathcal{S}_i \triangleleft \mathcal{C}_i$ , where  $\mathcal{S}_i \neq \{\perp\}$  and not performed by rule 1 ( $\mathcal{C}_i$  can be  $\{\perp\}$  but obtained in a further derivation step). So, for the rest of the demonstration we, assume the additional hypothesis:

*HH* all the the expressions  $\mathcal{S}_i$  are different from  $\{\perp\}$  and all the proofs  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  are performed by a rule different from rule 1.

Under these conditions, all the set expressions  $\mathcal{S}_i$  and  $\mathcal{S}$  must share the same external syntactical form (all they have are of the form  $\{\dots\}$  or  $f(\dots)$  or *fails*( $\dots$ ) or  $\bigcup_{X \in \dots} \dots$ ). Moreover, we will see that all the proofs  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  and  $\mathcal{S} \triangleleft \mathcal{C}$  must be done by the same rule of the calculus.

We proceed by simultaneous induction on the lengths (number of *SRL*-derivation steps)  $l_1, \dots, l_n$  of the proofs  $\mathcal{S}_1 \triangleleft \mathcal{C}_1, \dots, \mathcal{S}_n \triangleleft \mathcal{C}_n$ .

$l_1 = \dots = l_n = 1$  We have the following cases:

- some of them is of the form  $\{X\} \triangleleft \{X\}$ , by rule 2. Then it must be  $\mathcal{S}_i = \mathcal{S} = \{X\}$  for each  $i$  and, also by rule 2,  $\mathcal{C}_i = \{X\}$  for each  $i$ . For  $\mathcal{S}$  we can get  $\mathcal{C} = \{X\}$ , so clearly  $\mathcal{C}_i \sqsubseteq \mathcal{C}$  for each  $i$ .
- some of them is of the form  $\{c\} \triangleleft \{c\}$ , with  $c \in DC^0$  by rule 3. Then it must be  $\mathcal{S}_i = \mathcal{S} = \{c\}$ , and  $\mathcal{C}_i = \{c\}$  by rule 3. By the same rule we can get  $\mathcal{C} = \{c\}$  and clearly  $\mathcal{C}_i \sqsubseteq \mathcal{C}$  for each  $i$ .

- some proof is  $f(\bar{t}_i) \triangleleft \{\mathbb{F}\}$ , done by rule 5. This means that  $\bar{t}_i$  has a  $DC \cup \{\mathbb{F}\}$ -clash with all the heads of the rules of the program. As  $f(\bar{t}_i) \sqsubseteq \mathcal{S}$ ,  $\mathcal{S}$  has the form  $f(\bar{t})$ , with  $\bar{t}_i \sqsubseteq \bar{t}$ , what means that  $\bar{t}$  has the same clash. So, by the same rule 5 we can get the  $SAS \mathcal{C} = \{\mathbb{F}\}$  for  $\mathcal{S}$ .

On the other hand, for the rest of set-expressions  $\mathcal{S}_j$ , as  $\mathcal{S}_j \sqsubseteq \mathcal{S}$  it must be  $\mathcal{S}_j = f(\bar{t}_j)$ , with  $\bar{t}_j \sqsubseteq \bar{t}$ . We could have, in principle, two possibilities:

- the tuple  $\bar{t}_j$  has the same constructor symbol at the position of the clash. Then the clash is also present for it, and then as our program is *CIS*, rule 5 is the only applicable one, and  $\mathcal{C}_j = \{\mathbb{F}\}$ . Then  $\mathcal{C}_j \sqsubseteq \mathcal{C}$ .
- the tuple  $\bar{t}_j$  could have  $\perp$  at the position of the clash or some ancestor position. In this case, it is not possible to use the same c-instance of a program rule. In fact, it is not possible to use any c-instance of the program rules: as the program is *CIS* all the heads of the rules have a (different) constructor symbol at such position. In this situation the *SAS* for  $f(\bar{t}_j)$  can only be the trivial obtained by rule 1, but this is a contradiction with *HH*.

$l_i = 1, l_j > 1$  The case such that the non-trivial proof  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  is done in 1 derivation steps, and  $\mathcal{S}_j \triangleleft \mathcal{C}_j$  is done in  $l_j > 1$  steps is not possible as consequence of the base case.

$l_1 > 1, \dots, l_n > 1$  We have the following cases:

- if some proof  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  is of the form  $\{c(\bar{t}_i)\} \triangleleft \mathcal{C}_i$  by rule 3, it must be  $\mathcal{S} = \{c(\bar{t})\}$  with  $\bar{t}_i \sqsubseteq \bar{t}$ . For the rest of set-expressions, as  $\mathcal{S}_j \sqsubseteq \mathcal{S}$  it must be  $\mathcal{S}_j = \{c(\bar{t}_j)\}$  with  $\bar{t}_j \sqsubseteq \bar{t}$ . Then, for the set-expressions corresponding to the first elements of the tuples we have  $\{t_{11}\}, \dots, \{t_{1n}\} \sqsubseteq \{t_1\}$  and similar for rest of elements. By i.h. the *SAS*'s obtained for  $\{t_{11}\}, \dots, \{t_{1n}\}$  can be refined by a unique *SAS* obtained for  $\{t_1\}$ , and the same for the rest of elements of the tuples. So, clearly, by rule 3 we can get a *SAS*  $\mathcal{C}$  for  $c(\bar{t})$  such that  $\mathcal{C}_j \sqsubseteq \mathcal{C}$  for  $j \in \{1, \dots, n\}$ .
- if some of the proofs  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  is done by rule 4, then  $\mathcal{S}_i = f(\bar{t}_i)$  and the proof has the form:

$$\frac{\mathcal{S}'\theta_i \triangleleft \mathcal{C}_i}{f(\bar{t}_i) \triangleleft \mathcal{C}_i}$$

where  $f(\bar{u}) \rightarrow \mathcal{S}'$  is a program rule such that  $\bar{u}\theta_i = \bar{t}_i$ ,  $\theta_i \in CSubst_{\perp, \mathbb{F}}$ . As  $\mathcal{S}$  refines  $f(\bar{t}_i)$  it has to be  $\mathcal{S} = f(\bar{t})$  with  $\bar{t}_i \sqsubseteq \bar{t}$ . For the rest of proofs  $\mathcal{S}_j \triangleleft \mathcal{C}_j$ , the situation is similar to the case of a derivation by rule 4. It has to be  $\mathcal{S}_j = f(\bar{t}_j)$ , with  $\bar{t}_j \sqsubseteq \bar{t}$ . We have, in principle, two possibilities:

- the tuple  $\bar{t}_j$  is a c-instance of  $\bar{u}$ , i.e.,  $\bar{t}_j = \bar{u}\theta_j$ .
- the tuple  $\bar{t}_j$  is not a c-instance of  $\bar{u}$ . As  $\bar{t}_j \sqsubseteq \bar{t}$ , this means that  $\bar{t}_i$  differs from  $\bar{t}$  in that at some positions where the first has a non-trivial cterm, the second can have  $\perp$ . But, as the program is *CIS* the rest of heads of the rules have also constructor symbols at such position, what means that there is not any program rule applicable to proceed by rule 4, neither rule 5 ( $\perp$  do not produce clashes). Then the only possible proof for  $\mathcal{S}_j \triangleleft \mathcal{C}_j$  would be obtained by rule 1, what is a contradiction with *HH*.

At this point we have  $\mathcal{S}_j = f(\bar{t}_j) = f(\bar{u})\theta_j$  and  $\mathcal{S} = f(\bar{t}) = f(\bar{u})\theta$ , such that  $f(\bar{u})\theta_j \sqsubseteq f(\bar{u})\theta$ , for  $j \in \{1, \dots, n\}$ . This means that  $\theta_j \sqsubseteq \theta$  and then, for the c-instances of the body we have  $\mathcal{S}'\theta_i \sqsubseteq \mathcal{S}'\theta$  by Prop. 2. So, we are in i.h. and have a proof  $\mathcal{S}'\theta \triangleleft \mathcal{C}$  such that  $\mathcal{C}_j \sqsubseteq \mathcal{C}$  for  $j \in \{1, \dots, n\}$ . By rule 4, we can build a proof for  $\mathcal{S} \triangleleft \mathcal{C}$ .

- if some of the proofs  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  is done by rule 6, then  $\mathcal{S}_i = \text{fails}(\mathcal{S}'_i)$  and such proof has the form:

$$\frac{\mathcal{S}'_i \triangleleft \{\mathbb{F}\}}{\text{fails}(\mathcal{S}'_i) \triangleleft \{\text{true}\}}$$

Then it has to be  $\mathcal{S} = \text{fails}(\mathcal{S}')$  with  $\mathcal{S}'_i \sqsubseteq \mathcal{S}'$ . By i.h.  $\mathcal{S}' \triangleleft \{\mathbb{F}\}$ , so we can get  $\mathcal{C} = \{\text{true}\}$  by rule 6. The rest of  $\mathcal{S}_j$ 's must be of the  $\text{fails}(\mathcal{S}'_j)$ , with  $\mathcal{S}'_j \sqsubseteq \mathcal{S}'$ . By i.h. we have  $\mathcal{C}'_j \sqsubseteq \{\mathbb{F}\}$ . Then it must be  $\mathcal{C}'_j = \{\mathbb{F}\}$ , what implies that the only non trivial proof for  $\text{fails}(\mathcal{S}'_j)$  provides the  $\text{SAS } \mathcal{C}_j = \{\text{true}\}$  and clearly  $\mathcal{C}_j \sqsubseteq \mathcal{C}$  for  $j \in \{1, \dots, n\}$ .

- if some of the proofs  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  is done by rule 7, then  $\mathcal{S}_i = \text{fails}(\mathcal{S}'_i)$  and the proof has the form:

$$\frac{\mathcal{S}'_i \triangleleft \mathcal{C}'_i}{\text{fails}(\mathcal{S}'_i) \triangleleft \{\text{false}\}}$$

where  $\mathcal{C}'_i$  contains some  $t \notin \{\perp, \mathbb{F}\}$ . It has to be  $\mathcal{S} = \text{fails}(\mathcal{S}')$  with  $\mathcal{S}'_i \sqsubseteq \mathcal{S}'$ . By i.h.  $\mathcal{S}' \triangleleft \mathcal{C}'$ , with  $\mathcal{C}'_i \sqsubseteq \mathcal{C}'$ . Then  $\mathcal{C}'$  contains a refinement  $t'$  of  $t$ , i.e.,  $t' \notin \{\perp, \mathbb{F}\}$ . So, for  $\text{fails}(\mathcal{S})$  we can obtain the  $\text{SAS } \mathcal{C} = \{\text{false}\}$  by rule 7.

The rest of  $\mathcal{S}_j$ 's must be of the form  $\text{fails}(\mathcal{S}'_j)$ , with  $\mathcal{S}'_j \sqsubseteq \mathcal{S}'$ . By i.h. we have  $\mathcal{C}'_j \sqsubseteq \mathcal{C}'$ , and then, the proof for  $\text{fails}(\mathcal{S}'_j)$  must proceed by rule 7 and  $\mathcal{C}_j = \{\text{false}\}$ . So,  $\mathcal{C}_j \sqsubseteq \mathcal{C}$  for  $j \in \{1, \dots, n\}$

- if some of the proofs is done by rule 8, then  $\mathcal{S}_i = \bigcup_{X \in \mathcal{S}'_i} \mathcal{S}''_i$ , for  $i \in \{1, \dots, n\}$ , and  $\mathcal{S} = \bigcup_{X \in \mathcal{S}'} \mathcal{S}''$ , with  $\mathcal{S}'_i \sqsubseteq \mathcal{S}'$  and  $\mathcal{S}''_i \sqsubseteq \mathcal{S}''$ . Then, all the proofs must be done by rule 8. For instance, the proof for  $\mathcal{S}_i \triangleleft \mathcal{C}_i$  will have the form:

$$\frac{\mathcal{S}'_i \triangleleft \{t_{i1}, \dots, t_{im_i}\} \quad \mathcal{S}''_i[X/\{t_{i1}, \dots, t_{im_i}\}] \triangleleft \mathcal{C}_i}{\bigcup_{X \in \mathcal{S}'_i} \mathcal{S}''_i \triangleleft \mathcal{C}_i}$$

The trivial proofs for  $\mathcal{S}''_i[X/\{t_{i1}, \dots, t_{im_i}\}] \triangleleft \mathcal{C}_i$  produce the  $\text{SAS } \mathcal{C}_i = \{\perp\}$  refined by any  $\mathcal{C}$ . So, we focus on the non-trivial proofs for  $\mathcal{S}''_i[X/\{t_{i1}, \dots, t_{im_i}\}] \triangleleft \mathcal{C}_i$  that in general are done by rule 9' in the form:

$$\frac{\mathcal{S}''_i[X/t_{i1}] \triangleleft \mathcal{C}_{i1} \quad \dots \quad \mathcal{S}''_i[X/t_{im_i}] \triangleleft \mathcal{C}_{im_i}}{\mathcal{S}''_i[X/\{t_{i1}, \dots, t_{im_i}\}] \triangleleft \mathcal{C}_{i1} \cup \dots \cup \mathcal{C}_{im_i} = \mathcal{C}_i}$$

(If  $m_i = 1$  then rule 9' is not needed and the proof is simply  $\mathcal{S}''_i[X/\{t_{i1}\}] \triangleleft \mathcal{C}_i$ . The rest of the demonstration is the same).

In order to build the proof for  $\mathcal{S} \triangleleft \mathcal{C}$ , first by i.h. we can get  $\mathcal{S}' \triangleleft \{t_1, \dots, t_k\}$ , such that  $\{t_{i1}, \dots, t_{im_i}\} \sqsubseteq \{t_1, \dots, t_k\}$  for each  $i$  in  $\{1, \dots, n\}$ . Collecting together the terms  $t_{ij}$ , this is equivalent to say:  $\mathcal{S}' \triangleleft \{t_1, \dots, t_k\}$  such that  $\mathcal{C}' = \{t_{11}, \dots, t_{1m_1}, \dots, t_{n1}, \dots, t_{nm_n}\} \sqsubseteq \{t_1, \dots, t_k\}$

Now we apply succesively i.h. in the following way: we consider  $t_1$  and all the  $t_{ij} \in \mathcal{C}'$  such that  $t_{ij} \sqsubseteq t_1$ . By Prop. 3  $\mathcal{S}''_i[X/\{t_{ij}\}] \sqsubseteq \mathcal{S}''_i[X/\{t_1\}]$  for each of those  $t_{ij}$ . Then, by i.h. we can get  $\mathcal{S}''_i[X/\{t_1\}] \triangleleft \mathcal{C}'_1$  such that  $\mathcal{C}_{ij} \sqsubseteq \mathcal{C}'_1$ . And so on for  $t_2$  until  $t_k$ . Notice that in this process every  $t_{ij} \in \mathcal{C}'$  must be selected at least once, because  $\mathcal{C}' \sqsubseteq \{t_1, \dots, t_k\}$  what means that each  $t_{ij}$  must satisfy  $t_{ij} \sqsubseteq t_l$  for some  $l \in \{1, \dots, k\}$ .

Joining the proofs for  $\mathcal{S}[X/\{t_l\}] \triangleleft \mathcal{C}'_l$ , by rule 9' we obtain the proof  $\mathcal{S}''[X/\{t_1, \dots, t_k\}] \triangleleft \mathcal{C}'_1 \cup \dots \cup \mathcal{C}'_k$  and then by rule 8 we get  $\mathcal{S} \triangleleft \mathcal{C}'_1 \cup \dots \cup \mathcal{C}'_k = \mathcal{C}$  such that  $\mathcal{C}_1, \dots, \mathcal{C}_n \sqsubseteq \mathcal{C}$

The extension of this lemma to general set-expressions is not difficult but requires some reasoning.

**Theorem 1 (Refinement).** *Let  $\mathcal{S}_1, \dots, \mathcal{S}_n, \mathcal{S}$  be set-expressions such that  $\mathcal{S}_1 \sqsubseteq \mathcal{S}, \dots, \mathcal{S}_n \sqsubseteq \mathcal{S}$ . Then*

$$\mathcal{S}_1 \triangleleft \mathcal{C}_1, \dots, \mathcal{S}_n \triangleleft \mathcal{C}_n \Rightarrow \mathcal{S} \triangleleft \mathcal{C}$$

for some  $\mathcal{C}$  such that  $\mathcal{C}_1 \sqsubseteq \mathcal{C}, \dots, \mathcal{C}_n \sqsubseteq \mathcal{C}$ .

*Proof.* First assume the result proved for only two expressions,  $\mathcal{S}_1$  and  $\mathcal{S}_2$ . The extension to  $n$  set-expressions is direct:

- as  $\mathcal{S}_1 \sqsubseteq \mathcal{S}, \mathcal{S}_1 \triangleleft \mathcal{C}_1, \mathcal{S}_2 \sqsubseteq \mathcal{S}$  and  $\mathcal{S}_2 \triangleleft \mathcal{C}_2$  then we would have  $\mathcal{S} \triangleleft \mathcal{C}_{12}$  such that  $\mathcal{C}_1, \mathcal{C}_2 \sqsubseteq \mathcal{C}_{12}$ .
- as  $\mathcal{S} \sqsubseteq \mathcal{S}, \mathcal{S} \triangleleft \mathcal{C}_{12}, \mathcal{S}_3 \sqsubseteq \mathcal{S}$  and  $\mathcal{S}_3 \triangleleft \mathcal{C}_3$  then we would have  $\mathcal{S} \triangleleft \mathcal{C}_{123}$  such that  $\mathcal{C}_{12}, \mathcal{C}_3 \sqsubseteq \mathcal{C}_{123}$ .
- ...
- iterating this argument we would obtain  $\mathcal{S} \triangleleft \mathcal{C} = \mathcal{C}_{1\dots n}$  such that  $\mathcal{C}_1, \dots, \mathcal{C}_n \sqsubseteq \mathcal{C}$

So, it is enough to prove this result:

$$\left. \begin{array}{l} \mathcal{S}' \sqsubseteq \mathcal{S} \quad \mathcal{S}' \triangleleft \mathcal{C}' \\ \mathcal{S}'' \sqsubseteq \mathcal{S} \quad \mathcal{S}'' \triangleleft \mathcal{C}'' \end{array} \right\} \Rightarrow \mathcal{S} \triangleleft \mathcal{C} \text{ such that } \mathcal{C}', \mathcal{C}'' \sqsubseteq \mathcal{C}$$

In general, these set-expressions are unions of single set-expressions of the form:

$$\begin{aligned} \mathcal{S}' &= \mathcal{S}'_1 \cup \dots \cup \mathcal{S}'_n \\ \mathcal{S}'' &= \mathcal{S}''_1 \cup \dots \cup \mathcal{S}''_m \\ \mathcal{S} &= \mathcal{S}_1 \cup \dots \cup \mathcal{S}_k \end{aligned}$$

The *SAS*  $\mathcal{C}'$  for  $\mathcal{S}'$  is obtained (by rule 9' if  $n > 1$ ) joining the *SAS*'s for  $\mathcal{S}'_1, \dots, \mathcal{S}'_n$ , and similar for  $\mathcal{S}''$ . We can collect all the single set-expressions  $\mathcal{S}'_i$  and  $\mathcal{S}''_j$  such that  $\mathcal{S}'_i, \mathcal{S}''_j \sqsubseteq \mathcal{S}_1$ . By Lemma 2 we can get a *SAS*  $\mathcal{C}_1$  for  $\mathcal{S}_1$  that refines those obtained from  $\mathcal{S}'_i$  and  $\mathcal{S}''_j$ . Now, we can repeat the process for  $\mathcal{S}_2$  and so on until  $\mathcal{S}_k$ . Notice that all the set-expressions  $\mathcal{S}'_i$  and  $\mathcal{S}''_j$  are refined by some  $\mathcal{S}_l$ , what means that every *SAS*'s obtained for them is refined by some *SAS* obtained for some  $\mathcal{S}_l, l \in \{1, \dots, k\}$ . Then, by rule 9' we can get the *SAS*  $\mathcal{C} = \mathcal{C}_1 \cup \dots \cup \mathcal{C}_n$  for  $\mathcal{S}$  such that  $\mathcal{C}', \mathcal{C}'' \sqsubseteq \mathcal{C}$ .

Now, the results of consistency and monotonicity are easily proved from the previous theorem.

**Proposition 5 (Consistency).** *If  $\mathcal{S}$  and  $\mathcal{S}'$  are consistent and  $\mathcal{S} \triangleleft \mathcal{C}, \mathcal{S}' \triangleleft \mathcal{C}'$ , then  $\mathcal{C}$  and  $\mathcal{C}'$  are also consistent.*

*Proof.* If  $\mathcal{S}$  and  $\mathcal{S}'$  are consistent then there exist  $\mathcal{S}''$  such that  $\mathcal{S}, \mathcal{S}' \sqsubseteq \mathcal{S}''$ . Then, if  $\mathcal{S} \triangleleft \mathcal{C}, \mathcal{S}' \triangleleft \mathcal{C}'$ , by Th. 1  $\mathcal{S}'' \triangleleft \mathcal{C}''$  such that  $\mathcal{C}, \mathcal{C}' \sqsubseteq \mathcal{C}''$ , what means that  $\mathcal{C}$  and  $\mathcal{C}'$  are consistent.

As a consequence of this property we have consistency of failure: if  $\mathcal{S} \triangleleft \{\mathbb{F}\}$ , then  $\llbracket \mathcal{S} \rrbracket = \{\{\perp\}, \{\mathbb{F}\}\}$ . Another interesting property of the rewriting logic is monotonicity.

**Proposition 6 (Monotonicity).** *If  $\mathcal{S} \sqsubseteq \mathcal{S}'$  and  $\mathcal{S} \triangleleft \mathcal{C}$ , then  $\mathcal{S}' \triangleleft \mathcal{C}'$  with  $\mathcal{C} \sqsubseteq \mathcal{C}'$ .*

*Proof.* Trivial from Th. 1.

The following corollary claims that if we can get two *SAS*'s for the same set-expression, then we can get a third *SAS* that refines both.

**Corollary 3.** *If  $\mathcal{S} \triangleleft \mathcal{C}$  and  $\mathcal{S} \triangleleft \mathcal{C}'$  then  $\mathcal{S} \triangleleft \mathcal{C}''$  such that  $\mathcal{C}, \mathcal{C}' \sqsubseteq \mathcal{C}''$*

## 4 Operational Semantics

The narrowing relation to be defined in 4.2 makes an extensive use of a particular notion of context. The next section is devoted to formalize contexts in our framework and some other related aspects.

### 4.1 Contexts

A *context* is a set-expression with some holes in the places of subexpressions. Syntactically a context is:

$$C ::= \mathcal{S} \mid [] \mid \text{fails}(C_1) \mid \bigcup_{X \in C_1} C_2 \mid C_1 \cup C_2$$

where  $\mathcal{S}$  is a set-expression, and  $C_1$  and  $C_2$  are contexts. A *principal context* is defined as:

$$C ::= f(\bar{t}) \mid [] \mid \text{fails}(\mathcal{S}) \mid \bigcup_{X \in \mathcal{S}} C_1 \mid C_1 \cup C_2$$

where  $\mathcal{S}$  is a set-expression, and  $C_1$  and  $C_2$  are principal contexts. Intuitively, a principal context is a context without indexed holes, i.e., it has all its holes at the highest level.

Analogous to the case of set-expressions, we consider both *total (principal) contexts* when  $\mathcal{S} \in \text{SetExp}$  and *partial (principal) contexts* if  $\mathcal{S} \in \text{SetExp}_\perp$ . From the definitions of produced variables and free variables of a set-expression it is direct to define *produced variables*  $PV(C)$  and *free variables*  $FV(C)$  of a context  $C$ . As in the case of set-expressions, for contexts we also impose *admissibility conditions*: contexts of the form  $\bigcup_{X \in C_1} C_2$  must satisfy  $X \notin \text{var}(C_1) \cup PV(C_2)$  and  $PV(C_1) \cap PV(C_2) = \emptyset$ ; and contexts of the form  $C_1 \cup C_2$  must satisfy  $PV(C_1) \cap FV(C_2) = \emptyset$  and  $PV(C_2) \cap FV(C_1) = \emptyset$ . How to apply substitutions to contexts is defined in a natural way.

For any context  $C$  we can define the *arity* of  $C$ , namely  $|C|$ , as the number of holes of it; formally:

- $|\mathcal{S}| = 0$
- $|[]| = 1$
- $|\text{fails}(C_1)| = |C_1|$
- $|\bigcup_{X \in C_1} C_2| = |C_1| + |C_2|$
- $|C_1 \cup C_2| = |C_1| + |C_2|$

A context  $C$  of arity  $n$  can be understood as a function that takes  $n$  contexts  $C_1, \dots, C_n$  as arguments and returns another context resulting of fulfilling the holes with the contexts of the arguments. Formally, the *application of a context*  $C$  to the tuple of arguments  $C_1, \dots, C_n$ , notated as  $C [C_1, \dots, C_n]$ , is defined as:

- $[\ ] [C] = C$
- $fails(C) [C_1, \dots, C_n] = fails(C [C_1, \dots, C_n])$
- $(\bigcup_{X \in C} C') [C_1, \dots, C_n, C'_1, \dots, C'_m] = \bigcup_{X \in C} [C_1, \dots, C_n] C' [C'_1, \dots, C'_m]$ , where  $|C| = n$  and  $|C'| = m$
- $(C \cup C') [C_1, \dots, C_n, C'_1, \dots, C'_m] = (C [C_1, \dots, C_n]) \cup (C' [C'_1, \dots, C'_m])$ , where  $|C| = n$  and  $|C'| = m$

In the rest of the paper all the context applications are done in such a way that admissibility conditions are satisfied by the resulting context.

Notice that the application of a context  $C$  of arity  $n$  to a tuple of contexts  $C_1, \dots, C_n$  of arities  $m_1, \dots, m_n$  results in another context of arity  $m_1 + \dots + m_n$ . And of course, if we apply this context  $C$  to a tuple of set-expressions (contexts with arity 0) the resulting context has arity 0, that is, a set-expression. With respect to substitutions it is easy to check that for any  $\sigma \in CSubst_{\perp, F}$  we have  $(C [\mathcal{S}_1, \dots, \mathcal{S}_n])\sigma = C\sigma [\mathcal{S}_1\sigma, \dots, \mathcal{S}_n\sigma]$ .

Given a set-expression  $\mathcal{S}$ , it is possible to determine a principal context  $C_{\mathcal{S}}$  and a tuple of cterms  $t_1, \dots, t_n$  such that  $\mathcal{S} = C_{\mathcal{S}} [\{t_1\}, \dots, \{t_n\}]$ . We can see it reasoning on the structure of  $\mathcal{S}$ :

- if  $\mathcal{S} = \{t\}$ , then clearly  $\mathcal{S} = [\ ] [\{t\}]$
- if  $\mathcal{S} = f(\bar{t})$  or  $\mathcal{S} = fails(\mathcal{S}')$ , then  $C_{\mathcal{S}} = \mathcal{S}$  (a 0-arity principal context) and the tuple of arguments is empty.
- if  $\mathcal{S} = \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2$ , we have  $\mathcal{S}_2 = C_{\mathcal{S}_2} [\{t_1\}, \dots, \{t_n\}]$  (where  $C_{\mathcal{S}_2}$  is a principal context). Then, clearly  $\mathcal{S} = (\bigcup_{X \in \mathcal{S}_1} C_{\mathcal{S}_2}) [\{t_1\}, \dots, \{t_n\}]$ , and  $(\bigcup_{X \in \mathcal{S}_1} C_{\mathcal{S}_2})$  is a principal context.
- if  $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ , then if  $\mathcal{S}_1 = C_{\mathcal{S}_1} [\{t_1\}, \dots, \{t_n\}]$  and  $\mathcal{S}_2 = C_{\mathcal{S}_2} [\{s_1\}, \dots, \{s_m\}]$ , we have  $\mathcal{S} = (C_{\mathcal{S}_1} \cup C_{\mathcal{S}_2}) [\{t_1\}, \dots, \{t_n\}, \{s_1\}, \dots, \{s_m\}]$ .

We say that  $C_{\mathcal{S}} [\{t_1\}, \dots, \{t_n\}]$  is the *principal contextual form* (in short p.c.f) of  $\mathcal{S}$ . In the following, we frequently refer to a set-expression by its p.c.f. Moreover, as an abuse of notation we will omit the braces '{', '}' of the arguments and write  $C_{\mathcal{S}} [t_1, \dots, t_n]$  or simply  $C_{\mathcal{S}} [\bar{t}]$ .

Notice that the equality  $\mathcal{S} = C_{\mathcal{S}} [t_1, \dots, t_n]$ , viewed as a context application is a trivial identity. The important point of such format is that the cterms  $t_1, \dots, t_n$  reflect the skeleton or constructed part of the set-expression. With this idea, if  $C_{\mathcal{S}} [t_1, \dots, t_n]$  is the p.c.f. for  $\mathcal{S}$ , the *information set*  $\mathcal{S}^*$  of  $\mathcal{S}$  is defined as:

- $\{\perp\}$  if  $n = 0$ ;
- $\{t_1\tau, \dots, t_n\tau\}$  if  $n > 0$  where  $\tau$  is defined as  $X\tau = \perp$  if  $X \in PV(\mathcal{S})$  and  $X\tau = X$  otherwise

For example, consider the set-expression introduced in Sect. 3

$$\mathcal{S} = \bigcup_{A \in \bigcup_{B \in f(X, Y)} \{B\}} \{c(A, X)\} \cup \bigcup_{C \in \{X\}} f(C, Y)$$

The context  $C_{\mathcal{S}}$  of the p.c.f. only has a hole in the place of  $\{c(A, X)\}$ , which is the only argument of the p.c.f., that is:

$$(\bigcup_{A \in \bigcup_{B \in f(X, Y)} \{B\}} [\ ] \cup \bigcup_{C \in \{X\}} f(C, Y)) [c(A, X)]$$

The information set in this case is  $\mathcal{S}^* = \{c(\perp, X)\}$ , obtained by replacing the produced variable  $A$  by  $\perp$ .

Notice that, in the second case of the definition of information set,  $\tau$  replaces all the produced variables of  $t_1, \dots, t_n$  by  $\perp$  in such a way that  $\text{var}(\mathcal{S}^*) \cap \text{PV}(\mathcal{S}) = \emptyset$ . The information set  $\mathcal{S}^*$  is an special SAS for  $\mathcal{S}$ , in the sense that it can be obtained without evaluating any function call. This is what shows the following property.

**Proposition 7.** *The information set of a set-expression is a SAS for  $\mathcal{S}$ , i.e.,  $\mathcal{S} \triangleleft \mathcal{S}^*$ . Moreover, this SAS can be obtained by using rules 1, 2, 3, 8 and 9 of SRL (in other words, no function rule is needed to obtain  $\mathcal{S}^*$ ).*

*Proof.* Let  $C [t_1, \dots, t_n]$  be the p.c.f. for  $\mathcal{S}$ . We must prove  $C [t_1, \dots, t_n] \triangleleft \{t_1\tau, \dots, t_n\tau\}$ , where  $\tau$  replaces the produced variables of  $C$  by  $\perp$ . We proceed by induction on the structure of  $\mathcal{S}$ :

- If  $\mathcal{S} = \{t\}$  then its p.c.f. is  $[\ ] \{t\}$  and by 2  $\mathcal{S} \triangleleft \{t\}$ .
- The p.c.f. for  $\mathcal{S} = f(\bar{s})$  and  $\mathcal{S} = \text{fails}(\mathcal{S})$  is  $\mathcal{S}$  itself and  $\mathcal{S}^* = \{\perp\}$ . Clearly  $\mathcal{S} \triangleleft \{\perp\}$  by rule 1 of SRL.
- If  $\mathcal{S} = \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2$ , its principal contextual form is  $(\bigcup_{X \in \mathcal{S}_1} C_2) [t_1, \dots, t_2]$  where  $C_2 [t_1, \dots, t_2]$  is the p.c.f. for  $\mathcal{S}_2$ . By rule 8 we can build a proof of the form:

$$\frac{\mathcal{S}_1 \triangleleft \{\perp\} \quad (C_2 [t_1, \dots, t_2])[X/\perp] \triangleleft \mathcal{C}}{\bigcup_{X \in \mathcal{S}} (C_2 [t_1, \dots, t_2]) \triangleleft \mathcal{C}}$$

It is easy to see that  $(C_2 [t_1, \dots, t_2])[X/\perp] = C_2[X/\perp] [t_1[X/\perp], \dots, t_2[X/\perp]]$  and this is in fact a p.c.f. whose information set is the corresponding to the tuple  $[t_1[X/\perp], \dots, t_2[X/\perp]]$ . Then, by i.h. we can build a proof for

$$C_2[X/\perp] [t_1[X/\perp], \dots, t_2[X/\perp]] \triangleleft \{t_1[X/\perp]\tau', \dots, t_n[X/\perp]\tau'\}$$

where  $\tau = [X/\perp]\tau'$ . Then we can get  $\mathcal{C} = \{t_1\tau, \dots, t_n\tau\}$ , i.e.,  $(\bigcup_{X \in \mathcal{S}} (C_2 [t_1, \dots, t_2]) \triangleleft \{t_1\tau, \dots, t_n\tau\})$

- If  $\mathcal{S} = s_{-1} \cup \mathcal{S}_2$ , its principal contextual form is  $(C_{\mathcal{S}_1} \cup C_{\mathcal{S}_2}) [\{t_1\}, \dots, \{t_n\}, \{s_1\}, \dots, \{s_m\}]$ , where  $C_{\mathcal{S}_1} [\{t_1\}, \dots, \{t_n\}]$  and  $C_{\mathcal{S}_2} [\{s_1\}, \dots, \{s_m\}]$  are the p.c.f.'s for  $\mathcal{S}_1$  and  $\mathcal{S}_2$  respectively. On the other hand,  $\mathcal{S}^* = \{t_1\tau, \dots, t_n\tau, s_1\tau, \dots, s_m\tau\}$ . The result easily follows from rule 9' and i.h.

Another interesting property of principal contexts is that, as functions, they are compositional in the sense of the following property:

**Proposition 8 (Composition of Principal Contexts).** *Let  $C$  be a principal context of arity  $n$  and  $\mathcal{S}_1 = C_{\mathcal{S}_1} [\bar{s}_1], \dots, \mathcal{S}_n = C_{\mathcal{S}_n} [\bar{s}_n]$  p.c.f.'s. Then we have:*

$$C [C_{\mathcal{S}_1} [\bar{s}_1], \dots, C_{\mathcal{S}_n} [\bar{s}_n]] = (C [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]) [\bar{s}_1 \dots \bar{s}_n]$$

where  $C [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]$  is the p.c.f. for the resulting set-expression.

*Proof.* By induction on the structure of  $C$ :

- the cases  $f(\bar{t})$  and  $\text{fails}(\mathcal{S})$  are trivial because the arity  $n$  is 0;
- for the case  $C = [\ ]$ , by definition of context application,  $[\ ] (C_{\mathcal{S}_1} [\bar{s}_1]) = C_{\mathcal{S}_1} [\bar{s}_1] = ([\ ] [C_{\mathcal{S}_1}]) [\bar{s}_1]$

– for the case  $C = (\bigcup_{X \in \mathcal{S}'} C_1)$  we have:

$$\begin{aligned}
C [C_{\mathcal{S}_1} [\bar{s}_1], \dots, C_{\mathcal{S}_n} [\bar{s}_n]] &= \text{definition of } C \\
(\bigcup_{X \in \mathcal{S}'} C_1) [C_{\mathcal{S}_1} [\bar{s}_1], \dots, C_{\mathcal{S}_n} [\bar{s}_n]] &= \text{i.h.} \\
\bigcup_{X \in \mathcal{S}'} ((C_1 [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]) [\bar{s}_1, \dots, \bar{s}_n]) &= \text{context application} \\
(\bigcup_{X \in \mathcal{S}'} (C_1 [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]) [\bar{s}_1, \dots, \bar{s}_n]) &= \text{context application} \\
((\bigcup_{X \in \mathcal{S}'} C_1) [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]) [\bar{s}_1, \dots, \bar{s}_n] &= \text{definition of } C \\
(C [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]) [\bar{s}_1, \dots, \bar{s}_n] &
\end{aligned}$$

– for the case  $C = C_1 \cup C_2$ :

$$\begin{aligned}
C [C_{\mathcal{S}_1} [\bar{s}_1], \dots, C_{\mathcal{S}_n} [\bar{s}_n], C'_{\mathcal{S}'_1} [\bar{s}'_1], \dots, C_{\mathcal{S}'_m} [\bar{s}'_m]] &= \text{def. of } C \\
(C_1 \cup C_2) [C_{\mathcal{S}_1} [\bar{s}_1], \dots, C_{\mathcal{S}_n} [\bar{s}_n], C'_{\mathcal{S}'_1} [\bar{s}'_1], \dots, C_{\mathcal{S}'_m} [\bar{s}'_m]] &= \text{cntx. ap.} \\
(C_1 [C_{\mathcal{S}_1} [\bar{s}_1], \dots, C_{\mathcal{S}_n} [\bar{s}_n]]) \cup (C_2 [C'_{\mathcal{S}'_1} [\bar{s}'_1], \dots, C_{\mathcal{S}'_m} [\bar{s}'_m]]) &= \text{i.h.} \\
(C_1 [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}]) [\bar{s}_1, \dots, \bar{s}_n] \cup (C_2 [C'_{\mathcal{S}'_1}, \dots, C_{\mathcal{S}'_m}]) [\bar{s}'_1, \dots, \bar{s}'_m] &= \text{cntx. ap.} \\
((C_1 \cup C_2) [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}, C'_{\mathcal{S}'_1}, \dots, C_{\mathcal{S}'_m}]) [\bar{s}_1, \dots, \bar{s}_n, \bar{s}'_1, \dots, \bar{s}'_m] &= \text{def. of } C \\
(C [C_{\mathcal{S}_1}, \dots, C_{\mathcal{S}_n}, C'_{\mathcal{S}'_1}, \dots, C_{\mathcal{S}'_m}]) [\bar{s}_1, \dots, \bar{s}_n, \bar{s}'_1, \dots, \bar{s}'_m] &
\end{aligned}$$

## 4.2 A Narrowing Relation for Set-Expressions

The rewriting logic *SRL* showed in Sect. 3 fixes the denotational semantics of set-expressions. Now we are interested in a operational mechanism for narrowing set-expressions into simplified or normal forms, while finding appropriate values for the variables of the expressions.

The syntactic structure of a *normal form* is:

$$\{t_1\} \cup \dots \cup \{t_n\}, \text{ with } t_1, \dots, t_n \in CTerm_{\mathcal{F}}.$$

According to this,  $\{\mathcal{F}\}$  is a normal form itself. Notice also that a normal form cannot contain any function symbol or indexed union. On the other hand, the undefined value  $\perp$  has not any sense in the operational mechanism that deals only with total set-expressions.

**One-narrowing-step relation:** Given a program  $\mathcal{P}$ , two set-expressions  $\mathcal{S}, \mathcal{S}' \in SetExp$ ,  $\theta \in CSubst_{\mathcal{F}}$  and  $\Gamma \subseteq \mathcal{V}$ , a narrowing step is expressed as:

$$\mathcal{S} \xrightarrow[\theta]{\Gamma} \mathcal{S}'$$

where the relation  $\mathcal{S} \xrightarrow[\theta]{\Gamma} \mathcal{S}'$  is defined in the Table 2. The set  $\Gamma$  is the *set of protected variables*: produced variables of the external scope are stored into  $\Gamma$  in order to avoid to apply substitutions for them. For narrowing a set-expression  $\mathcal{S}$  we must take  $\Gamma$  such that  $\Gamma \cap FV(\mathcal{S}) = \emptyset$ ; in fact, the most simple choice is  $\Gamma = \emptyset$ . The calculus itself will protect the appropriate variables in derivations.

In the following we use *SNarr* as a name for this narrowing relation, and refer to it as *set-narrowing*.

Some comments about the rules in Table 2.:

<b>Cntx</b>	$C[S] \xrightarrow[\theta]{\Gamma} C\theta[S']$	if $S \xrightarrow[\theta]{\Gamma \cup FV(C)} S'$
<b>Nrrw<sub>1</sub></b>	$f(\bar{t}) \xrightarrow[\theta _{\text{var}(\bar{t})}]{\Gamma} S\theta$	if $(f(\bar{s}) \rightarrow S) \in \mathcal{P}$ , $\theta \in CSubst_{\mathbb{F}}$ is a m.g.u. for $\bar{s}$ and $\bar{t}$ with $Dom(\theta) \cap \Gamma = \emptyset$
<b>Nrrw<sub>2</sub></b>	$f(\bar{t}) \xrightarrow[\epsilon]{\Gamma} \{\mathbb{F}\}$	if for every rule $(f(\bar{s}) \rightarrow S_1) \in \widehat{\mathcal{P}}$ , $\bar{s}$ and $\bar{t}$ have a $DC \cup \{\mathbb{F}\}$ -clash
<b>Fail<sub>1</sub></b>	$fails(S) \xrightarrow[\epsilon]{\Gamma} \{true\}$	if $S^* = \{\mathbb{F}\}$
<b>Fail<sub>2</sub></b>	$fails(S) \xrightarrow[\epsilon]{\Gamma} \{false\}$	if $\exists t \in S^* t \neq \perp, t \neq \mathbb{F}$
<b>Flat</b>	$\bigcup_{X \in \bigcup_{Y \in S_1} S_2} S_3 \xrightarrow[\epsilon]{\Gamma} \bigcup_{Y \in S_1} \bigcup_{X \in S_2} S_3$	
<b>Dist</b>	$\bigcup_{X \in S_1 \cup S_2} S_3 \xrightarrow[\epsilon]{\Gamma} \bigcup_{X \in S_1} S_3 \cup \bigcup_{X \in S_2} S_3$	
<b>Bind</b>	$\bigcup_{X \in \{t\}} S \xrightarrow[\epsilon]{\Gamma} S[X/t]$	
<b>Elim</b>	$\bigcup_{X \in S'} S \xrightarrow[\epsilon]{\Gamma} S$	if $X \notin FV(S)$

Table 2. *SNarr* relation

- **Cntx** performs a sub-derivation on a sub-set-expression and then replaces the original by the resulting one. Notice that it ensures the protection of produced variables of the context by adding them to  $\Gamma$ . It is not difficult to see that repeated applications of rule **Cntx** can always be simplified to a single application of it.

- The rule **Nrrw<sub>1</sub>** narrows a function call  $f(\bar{t})$  using a rule of the program. The m.g.u.  $\theta$  used for the parameter passing is not allowed to affect protected variables due to the condition  $Dom(\theta) \cap \Gamma = \emptyset$ . It is also clear that  $Ran(\theta) \cap \Gamma = \emptyset$  because the variables of the rule are fresh variables. Notice also that, for technical convenience, the substitution  $\theta$  is projected over the variables of the narrowed expression. Notice finally that this is *the* rule which really binds variables, since **Cntx** is merely a contextual rule and the rest produce an empty substitution.

- **Nrrw<sub>2</sub>** operates only in the case that **Nrrw<sub>1</sub>** cannot find an applicable rule of the program, i.e., all of them have a clash. As our programs are *DC-complete* this situation is only possible if the call  $f(\bar{t})$  has  $\mathbb{F}$  at a position in which the heads of program rules have a constructor symbol  $c \in DC$  (the transformation to *CIS-programs* does not introduce  $\mathbb{F}$  in heads).

- Rules **Fail<sub>1</sub>** and **Fail<sub>2</sub>** are direct counterpart of rules 6 and 7 of *SRL*.

- Rules **Flat**, **Dist**, **Bind** and **Elim** directly reflect some properties of sets in the mathematical sense. From an intuitive point of view, they all reduce the structural complexity of the set-expression.

The closure  $S \xrightarrow[\theta]{\Gamma}^* S'$  is defined in the usual way:

$$S = S_0 \xrightarrow[\theta_1]{\Gamma} S_1 \xrightarrow[\theta_2]{\Gamma} \dots \xrightarrow[\theta_n]{\Gamma} S_n = S', \quad \text{with } \theta = \theta_1 \theta_2 \dots \theta_n$$

It is easy to see that at any step  $S_i \theta_i = S_i$  and also  $S' \theta = S'$ .

**Example** consider the *CIS*-program of graphs of Sect. 3. The node  $c$  is safe according to definitions, so *SNarr* must be able to narrow the expression  $safe(X)$  to  $true$  with the substitution  $X = c$ . Below we show the steps of such derivation; we underline the redex at each step and annotate the rule of *SNarr* applied. The applications of **Cntx** are omitted for simplicity. We assume the function *ifThenElse* defined by the natural rules, and we shorten *ifThenElse* by *iTe*. We assume also that the function *eq* is defined by rules  $eq(a, a) \rightarrow \{true\}$ ,  $eq(a, b) \rightarrow \{false\}$  and so on. The answer substitution  $[X/c]$  is found in the third step by **Nrrw**<sub>1</sub> applied to the expression  $eq(X, d)$ .

<u><math>safe(X)</math></u>	(Nrrw <sub>1</sub> )
<u><math>fails(path(X, d))</math></u>	(Nrrw <sub>1</sub> )
<u><math>fails(\bigcup_{A \in eq(X, d)} \bigcup_{B \in \bigcup_{C \in next(X)} path(C, d)} iTe(A, true, B))</math></u>	<span style="border: 1px solid black; padding: 2px;"><math>X/c</math></span> (Nrrw <sub>1</sub> )
<u><math>fails(\bigcup_{A \in \{false\}} \bigcup_{B \in \bigcup_{C \in next(c)} path(C, d)} iTe(A, true, B))</math></u>	(Bind)
<u><math>fails(\bigcup_{B \in \bigcup_{C \in next(c)} path(C, d)} iTe(false, true, B))</math></u>	(Nrrw <sub>1</sub> )
<u><math>fails(\bigcup_{B \in \bigcup_{C \in next(c)} path(C, d)} \{B\})</math></u>	(Flat)
<u><math>fails(\bigcup_{C \in next(c)} \bigcup_{B \in path(C, d)} \{B\})</math></u>	(Nrrw <sub>1</sub> )
<u><math>fails(\bigcup_{C \in next(c)} \bigcup_{B \in \bigcup_{D \in eq(C, d)} \bigcup_{E \in \bigcup_{F \in next(C)} path(F, d)} iTe(D, true, E)} \{B\})</math></u>	(Flat)
<u><math>fails(\bigcup_{C \in next(c)} \bigcup_{D \in eq(C, d)} \bigcup_{B \in \bigcup_{E \in \bigcup_{F \in next(C)} path(F, d)} iTe(D, true, E)} \{B\})</math></u>	(Flat)
<u><math>fails(\bigcup_{C \in next(c)} \bigcup_{D \in eq(C, d)} \bigcup_{E \in \bigcup_{F \in next(C)} path(F, d)} \bigcup_{B \in iTe(D, true, E)} \{B\})</math></u>	(Nrrw <sub>2</sub> )
<u><math>fails(\bigcup_{C \in \{F\}} \bigcup_{D \in eq(C, d)} \bigcup_{E \in \bigcup_{F \in next(C)} path(F, d)} \bigcup_{B \in iTe(D, true, E)} \{B\})</math></u>	(Bind)
<u><math>fails(\bigcup_{D \in eq(F, d)} \bigcup_{E \in \bigcup_{F \in next(F)} path(F, d)} \bigcup_{B \in iTe(D, true, E)} \{B\})</math></u>	(Nrrw <sub>2</sub> )
<u><math>fails(\bigcup_{D \in \{F\}} \bigcup_{E \in \bigcup_{F \in next(F)} path(F, d)} \bigcup_{B \in iTe(D, true, E)} \{B\})</math></u>	(Bind)
<u><math>fails(\bigcup_{E \in \bigcup_{F \in next(F)} path(F, d)} \bigcup_{B \in iTe(F, true, E)} \{B\})</math></u>	(Nrrw <sub>2</sub> )
<u><math>fails(\bigcup_{E \in \bigcup_{F \in next(F)} path(F, d)} \bigcup_{B \in \{F\}} \{B\})</math></u>	(Elim)
<u><math>fails(\bigcup_{B \in \{F\}} \{B\})</math></u>	(Bind)
<u><math>fails(\{F\})</math></u>	(Fail <sub>1</sub> )
$\{true\}$	

**Table 3.** A narrowing derivation for  $safe(X)$

This derivation has been performed by a little prototype that implements the relation *SNarr* with an appropriate criterion for selecting the redex. In fact this prototype provides four possible reductions for the expression  $safe(X)$ :  $false$  with  $[X/a]$ ;  $false$  with  $[X/b]$ ;  $true$  with  $[X/c]$  and  $false$  with  $[X/d]$ .

A first easy property of the relation  $SNarr$  is that the resulting set-expression has already applied the substitution, and as it is idempotent we have:

**Proposition 9.** *If  $\mathcal{S} \xrightarrow[\theta]{F} \mathcal{S}'$  then  $\mathcal{S}'\theta = \mathcal{S}'$*

### 4.3 Correctness and completeness of $Snarr$

A desirable property is the preservation of semantics under context application: consider two set-expressions  $\mathcal{S}$  and  $\mathcal{S}'$  with  $\llbracket \mathcal{S} \rrbracket = \llbracket \mathcal{S}' \rrbracket$  and a 1-arity context  $C$ ; then we would like  $\llbracket C[\mathcal{S}] \rrbracket = \llbracket C[\mathcal{S}'] \rrbracket$ . But this property is not true in general. For example, assume a program with the rules  $f(z) \rightarrow \{z\}$  and  $g(z) \rightarrow \{s(z)\}$ . It is not difficult to see that for the set-expressions  $f(X)$  and  $g(X)$  we have  $\llbracket f(X) \rrbracket = \llbracket f(X) \rrbracket = \{\perp\}$ . But if we consider the context  $C = \bigcup_{X \in \{z\}} [ ]$  then we have  $C[f(X)] = \bigcup_{X \in \{z\}} f(X)$  and  $C[g(X)] = \bigcup_{X \in \{z\}} g(X)$ , whose denotations are respectively  $\{\{\perp\}, \{z\}\}$  and  $\{\{\perp\}, \{s(z)\}\}$ .

What is true is the following result:

**Lemma 3.** *Let  $\mathcal{S}$  and  $\mathcal{S}'$  be set-expressions such that  $\llbracket \mathcal{S}\sigma \rrbracket = \llbracket \mathcal{S}'\sigma \rrbracket$  for any  $\sigma \in CSubst_{\perp, F}$  admissible for  $\mathcal{S}$  and  $\mathcal{S}'$ . Then, for any context  $C$  of arity 1,  $\llbracket (C[\mathcal{S}])\sigma' \rrbracket = \llbracket (C[\mathcal{S}'])\sigma' \rrbracket$ , for any  $\sigma' \in CSubst_{\perp, F}$  admissible for  $C[\mathcal{S}]$  and  $C[\mathcal{S}']$ .*

Notice that in the example above, the hypothesis  $\llbracket g(X)\theta \rrbracket = \llbracket f(X)\theta \rrbracket$  does not hold, for instance taking  $\theta = [X/z]$ . Now we prove this result.

*Proof.* We must prove  $(C[\mathcal{S}])\sigma \triangleleft \mathcal{C} \Leftrightarrow (C[\mathcal{S}'])\sigma \triangleleft \mathcal{C}$ . We reason by induction on the structure of the context  $C$ . The possible forms for  $C$  with  $|C| = 1$  are:

- $C = [ ]$ : direct from the hypothesis.
- $C = fails(C_1)$ : direct from rules 6 and 7 of  $SRL$ .
- $C = \bigcup_{X \in C_1} \mathcal{S}_2$ . The proof for  $(C[\mathcal{S}])\sigma \triangleleft \mathcal{C}$  must be done by rule 8 of  $CRWL$  in the form:

$$\frac{(C_1[\mathcal{S}])\sigma \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2\sigma[X/C_1] \triangleleft \mathcal{C}}{\bigcup_{X \in (C_1[\mathcal{S}])\sigma} \mathcal{S}_2\sigma \triangleleft \mathcal{C}}$$

By i.h.  $(C_1[\mathcal{S}])\sigma \triangleleft \mathcal{C}_1 \Leftrightarrow (C_1[\mathcal{S}'])\sigma \triangleleft \mathcal{C}_1$  and then the proof for  $(\bigcup_{X \in C_1[\mathcal{S}']} \mathcal{S}_2)\sigma \triangleleft \mathcal{C}$  can also be done by rule 8 with the same form.

- $C = \bigcup_{X \in \mathcal{S}_1} C_1$ . The proof for  $(C[\mathcal{S}])\sigma \triangleleft \mathcal{C}$  must be done by rule 8 of  $CRWL$  in the form:

$$\frac{\mathcal{S}_1\sigma \triangleleft \{t_1, \dots, t_n\} \quad (C_1[\mathcal{S}])\sigma[X/t_1] \cup \dots \cup (C_1[\mathcal{S}])\sigma[X/t_n] \triangleleft \mathcal{C}'_1 \cup \dots \cup \mathcal{C}'_n}{\bigcup_{X \in \mathcal{S}_1\sigma} (C_1[\mathcal{S}])\sigma \triangleleft \mathcal{C}'_1 \cup \dots \cup \mathcal{C}'_n = \mathcal{C}}$$

where each  $\mathcal{C}'_i$  is a  $SAS$  for  $(C_1[\mathcal{S}])\sigma[X/t_i]$ . The substitutions  $\sigma_i = \sigma[X/t_i]$  are all in the i.h. and then  $(C_1[\mathcal{S}])\sigma[X/t_i] \triangleleft \mathcal{C}'_i \Leftrightarrow (C_1[\mathcal{S}'])\sigma[X/t_i] \triangleleft \mathcal{C}'_i$ . So, the proof for  $\bigcup_{X \in \mathcal{S}_1\sigma} (C_1[\mathcal{S}'])\sigma \triangleleft \mathcal{C}'_1 \cup \dots \cup \mathcal{C}'_n = \mathcal{C}$  has the same form.

- $C = C_1 \cup \mathcal{S}_2$  (the case  $C = \mathcal{S}_1 \cup C_2$  is similar): direct from rule 9 of  $CRWL$  and i.h.

Lemma 3 is used for proving the following correctness result:

**Theorem 2 (Correctness of  $SNarr$ ).** Let  $\mathcal{S}, \mathcal{S}' \in SetExp$ ,  $\theta \in CSubst_{\mathbb{F}}$ ,  $\Gamma \subseteq \mathcal{V}$  any set of variables, and  $\sigma \in CSubst_{\perp, \mathbb{F}}$  any admissible substitution for  $\mathcal{S}\theta$  and  $\mathcal{S}'$ . Then:

$$\mathcal{S} \xrightarrow[\theta]{\Gamma}^* \mathcal{S}' \Rightarrow \llbracket \mathcal{S}\theta\sigma \rrbracket = \llbracket \mathcal{S}'\sigma \rrbracket$$

In particular, taking  $\sigma = \epsilon$  we have:

$$\mathcal{S} \xrightarrow[\theta]{\Gamma}^* \mathcal{S}' \Rightarrow \llbracket \mathcal{S}\theta \rrbracket = \llbracket \mathcal{S}' \rrbracket$$

*Proof.* We consider a single derivation step  $\mathcal{S} \xrightarrow[\theta]{\Gamma}^* \mathcal{S}'$ ; the extension to  $n$  derivation steps comes from an easy induction on  $n$ .

**Nrrw<sub>1</sub>** Assume  $f(\bar{t}) \xrightarrow[\theta]{\Gamma} \mathcal{S}\theta$  using a rule  $f(\bar{s}) \rightarrow \mathcal{S}$  of the program such that  $\theta$  is a m.g.u. for  $\bar{s}$  and  $\bar{t}$  with  $Dom(\theta) \cap \Gamma = \emptyset$ . Then we have  $f(\bar{t})\theta = f(\bar{s})\theta$  and also  $f(\bar{t})\theta\sigma = f(\bar{s})\theta\sigma$  for any  $\sigma$ . By rule 4 of  $SRL$ ,  $f(\bar{t})\theta\sigma \triangleleft \mathcal{C}$  iff  $\mathcal{S}\theta\sigma \triangleleft \mathcal{C}$ , so it is clear that the denotation of  $f(\bar{t})\theta\sigma$  is the same as the denotation of  $\mathcal{S}\theta\sigma$ .

**Nrrw<sub>2</sub>** Assume  $f(\bar{t}) \xrightarrow[\epsilon]{\Gamma} \{\mathbb{F}\}$  and  $f(\bar{t})$  clash with all the heads of the rules of  $\mathcal{P}$ . Then  $\sigma$  preserves the clash and by rule 4 of  $CRWL$  we can get  $f(\bar{t})\sigma \triangleleft \{\mathbb{F}\}$ . In fact, by Prop. 5 this is the only possible non-trivial  $SAS$  for  $f(\bar{t})\sigma$ , the same as for  $\{\mathbb{F}\}$  ( $\sigma$  does not affect to  $\mathbb{F}$ ).

**Fail<sub>1</sub>** If  $fails(\mathcal{S}) \xrightarrow[\epsilon]{\Gamma} \{true\}$  and  $\mathcal{S}^* = \{\mathbb{F}\}$ , clearly  $(\mathcal{S}\sigma)^* = \{\mathbb{F}\}$  and by Prop. 7  $\mathcal{S}\sigma \triangleleft \{\mathbb{F}\}$ .

**Fail<sub>2</sub>** Assume  $fails(\mathcal{S}) \xrightarrow[\epsilon]{\Gamma} \{false\}$  and  $t \in \mathcal{S}^* - \{\perp, \mathbb{F}\}$ . Then the p.c.f. of  $\mathcal{S}$  must be of the form  $C_{\mathcal{S}} [\dots, t', \dots]$  with  $t'\tau = t$  ( $\tau$  replaces produced variables of  $C_{\mathcal{S}}$  by  $\perp$ ), and the p.c.f. for  $\mathcal{S}\sigma$  will have the form  $C_{\mathcal{S}\sigma} [\dots, t'\sigma, \dots]$ . The important fact is that  $t'\sigma$  cannot be  $\perp$ , neither  $\mathbb{F}$ , neither  $X \in PV(C)$ . By Prop. 7,  $\mathcal{S}\sigma \triangleleft \mathcal{C}$  with  $t'\sigma \in \mathcal{C}$ . Then by rule 7 of  $CRWL$  we obtain  $fails(\mathcal{S}\sigma) \triangleleft \{false\}$ . By Prop. 5 this is the only possible non-trivial  $SAS$  for  $fails(\mathcal{S}\sigma)$ , the same as for  $\{false\}$  ( $\sigma$  does not affect to  $false$ ).

**Flat** For a derivation of the form  $\bigcup_{X \in \bigcup_{Y \in \mathcal{S}_1} \mathcal{S}_2} \mathcal{S}_3 \xrightarrow[\epsilon]{\Gamma} \bigcup_{Y \in \mathcal{S}_1} \bigcup_{X \in \mathcal{S}_2} \mathcal{S}_3$ , we must prove:

$$\bigcup_{X \in \bigcup_{Y \in \mathcal{S}_1\sigma} \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C} \text{ iff } \bigcup_{Y \in \mathcal{S}_1\sigma} \bigcup_{X \in \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}.$$

First, the  $SAS \{\perp\}$  can be trivially obtained for both expressions. Let us see the form of a non-trivial derivation in  $CRWL$  for both expressions. For the first one we have, by rule 8 twice:

$$\frac{\frac{\mathcal{S}_1\sigma \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2\sigma[Y/C_1] \triangleleft \mathcal{C}_2}{\bigcup_{Y \in \mathcal{S}_1\sigma} \mathcal{S}_2\sigma \triangleleft \mathcal{C}_2} \quad \mathcal{S}_3\sigma[X/C_2] \triangleleft \mathcal{C}_3}{\bigcup_{X \in \bigcup_{Y \in \mathcal{S}_1\sigma} \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}_3}$$

While for the second expression we can get the same *SAS* also by rule 8 twice:

$$\frac{\mathcal{S}_1\sigma \triangleleft \mathcal{C}_1 \quad \frac{\mathcal{S}_2\sigma[Y/\mathcal{C}_1] \triangleleft \mathcal{C}_2 \quad \mathcal{S}_3\sigma[X/\mathcal{C}_2] \triangleleft \mathcal{C}_3}{(\bigcup_{X \in \mathcal{S}_2\sigma} \mathcal{S}_3\sigma)[Y/\mathcal{C}_1] \triangleleft \mathcal{C}_3}}{\bigcup_{Y \in \mathcal{S}_1\sigma} \bigcup_{X \in \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}_3}$$

**Dist** For a derivation of the form  $\bigcup_{X \in \mathcal{S}_1 \cup \mathcal{S}_2} \mathcal{S}_3 \xrightarrow[\epsilon]{\Gamma} \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_3 \cup \bigcup_{X \in \mathcal{S}_2} \mathcal{S}_3$ , again we study the form of a non-trivial derivation for both set-expressions in *SRL*. For the first one by rule 8 and then 9:

$$\frac{\frac{\mathcal{S}_1\sigma \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2\sigma \triangleleft \mathcal{C}_2}{\mathcal{S}_1\sigma \cup \mathcal{S}_2\sigma \triangleleft \mathcal{C}_1 \cup \mathcal{C}_2} \quad \mathcal{S}_3\sigma[X/\mathcal{C}_1 \cup \mathcal{C}_2] \triangleleft \mathcal{C}_{3,1-2}}{\bigcup_{X \in \mathcal{S}_1\sigma \cup \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}_{3,1-2}}$$

For the second expression we have by rule 9 and then 8 twice:

$$\frac{\frac{\mathcal{S}_1\sigma \triangleleft \mathcal{C}_1 \quad \mathcal{S}_3\sigma[X/\mathcal{C}_1] \triangleleft \mathcal{C}_{3,1}}{\bigcup_{X \in \mathcal{S}_1\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}_{3,1}} \quad \frac{\mathcal{S}_2\sigma \triangleleft \mathcal{C}_2 \quad \mathcal{S}_3\sigma[X/\mathcal{C}_2] \triangleleft \mathcal{C}_{3,2}}{\bigcup_{X \in \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}_{3,2}}}{\bigcup_{X \in \mathcal{S}_1\sigma} \mathcal{S}_3\sigma \cup \bigcup_{X \in \mathcal{S}_2\sigma} \mathcal{S}_3\sigma \triangleleft \mathcal{C}_{3,1} \cup \mathcal{C}_{3,2}}$$

It is easy to see to identify  $\mathcal{C}_{3,1-2} = \mathcal{C}_{3,1} \cup \mathcal{C}_{3,2}$ .

**Bind** Consider a derivation of the form  $\bigcup_{X \in \{t\}} \mathcal{S} \xrightarrow[\epsilon]{\Gamma} \mathcal{S}[X/t]$ . We must prove:

$$\bigcup_{X \in \{t\sigma\}} \mathcal{S}\sigma \triangleleft \mathcal{C} \text{ iff } \mathcal{S}[X/t]\sigma \triangleleft \mathcal{C}$$

Of course, it must be  $X \notin \text{Dom}(\sigma)$  in order to apply to the left expression. Then as it also is idempotent, the right proof is equivalent to  $\mathcal{S}\sigma[X/t\sigma] \triangleleft \mathcal{C}$ .

Apart from the trivial proof, a proof for the left hand side must be done by rule 8:

$$\frac{\{t\sigma\} \triangleleft \mathcal{C}' \quad \mathcal{S}\sigma[X/\mathcal{C}'\sigma] \triangleleft \mathcal{C}}{\bigcup_{X \in \{t\sigma\}} \mathcal{S}\sigma \triangleleft \mathcal{C}}$$

Now for proving the left to right implication, by Prop. 2 it must be  $\mathcal{C}' = \{t'\sigma\}$  with  $t'\sigma \sqsubseteq t\sigma$ . Then  $\mathcal{S}\sigma[X/\mathcal{C}'\sigma] = \mathcal{S}\sigma[X/t'\sigma]$  and  $[X/t'\sigma] \sqsubseteq [X/t\sigma]$ . By Lemma 1 we have  $\mathcal{S}\sigma[X/t\sigma] \triangleleft \mathcal{C}$ .

For the left to right implication, by Prop. 2  $\{t\sigma\} \triangleleft \{t\sigma\}$  and as  $\mathcal{S}\sigma[X/t\sigma] \triangleleft \mathcal{C}$ , by rule 8 we can build the proof:

$$\frac{\{t\sigma\} \triangleleft \{t\sigma\} \quad \mathcal{S}\sigma[X/t\sigma] \triangleleft \mathcal{C}}{\bigcup_{X \in \{t\sigma\}} \mathcal{S}\sigma \triangleleft \mathcal{C}}$$

**Elim** If  $\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2 \xrightarrow[\epsilon]{\Gamma} \mathcal{S}$ , having  $X \notin \text{FV}(\mathcal{S})$ , then by rule 8 of *CRWL* the denotation of  $\bigcup_{X \in \mathcal{S}_1\sigma} \mathcal{S}_2\sigma$  is clearly the same as the denotation of  $\mathcal{S}_2\sigma$  because the substitution  $[X/\dots]$  has no effect.

**Cntx** If  $C [S] \xrightarrow[\theta]{\Gamma} C\theta [S']$ , where  $S \xrightarrow[\theta]{\Gamma \cup PV(C)} S'$ . By i.h.  $\llbracket S\theta\sigma \rrbracket = \llbracket S'\sigma \rrbracket$  for any  $\sigma$  admissible for  $S\theta$  and  $S'$ . Then by Lemma 3  $\llbracket (C\theta [S\theta])\mu \rrbracket = \llbracket (C\theta [S'])\mu \rrbracket$ , for any  $\mu$  admissible for  $C\theta [S\theta]$  and  $C\theta [S']$ .

In order to prove completeness we need some previous results:

**Proposition 10 (Flatten).** *Let  $C$  be a principal context of arity  $n$ . Then it is possible to derive:*

$$\bigcup_{X \in C[S_1, \dots, S_n]} S \xrightarrow[\epsilon]{\Gamma} C [\bigcup_{X \in S_1} S, \dots, \bigcup_{X \in S_n} S]$$

using only **Ctnx**, **Flat** and **Dist**

*Proof.* We proceed by induction on the structure of the context  $C$ :

- If the context  $C$  is a set-expression  $S'$  then the result is trivial in 0 derivation steps.
- If we have  $\bigcup_{X \in [\ ] [S_1]} S$  then as it is the same as  $[\ ] \bigcup_{X \in S_1} S$  again in 0 derivation steps we have the result.
- If we have  $\bigcup_{X \in \bigcup_{Y \in S'} C' [S_1, \dots, S_n]} S$  then by rule **Flat** we can derive

$$\bigcup_{Y \in S'} \bigcup_{X \in C' [S_1, \dots, S_n]} S. \text{ By } \mathbf{Ctnx} \text{ and the i.h. we can derive}$$

$$\bigcup_{Y \in S'} C' [\bigcup_{X \in S_1} S, \dots, \bigcup_{X \in S_n} S].$$

- For the case  $\bigcup_{X \in \underbrace{C_1 \cup C_2}_C [S_1, \dots, S_n, S_{n+1}, \dots, S_m]} S$ , it is equivalent to  $\bigcup_{X \in C_1 [S_1, \dots, S_n] \cup C_2 [S_{n+1}, \dots, S_m]} S$  (assuming  $|C_1| = n$ ,  $|C_2| = m - n + 1$ ). By **Dist** we obtain  $\bigcup_{X \in C_1 [S_1, \dots, S_n]} S \cup \bigcup_{X \in C_2 [S_{n+1}, \dots, S_m]} S$ . Now, by **Ctnx** and the i.h. we get  $C_1 [\bigcup_{X \in S_1} S, \dots, \bigcup_{X \in S_n} S] \cup C_2 [\bigcup_{X \in S_{n+1}} S, \dots, \bigcup_{X \in S_m} S]$  which is in fact equivalent to  $\underbrace{C_1 \cup C_2}_C [\bigcup_{X \in S_1} S, \dots, \bigcup_{X \in S_n} S, \bigcup_{X \in S_{n+1}} S, \dots, \bigcup_{X \in S_m} S]$

**Corollary 4 (Flatten).** *Let  $C_S [\{t_1\}, \dots, \{t_n\}]$  the p.c.f. of a set-expression  $S$ . Then it is possible to derive:*

$$\bigcup_{X \in C_S [\{t_1\}, \dots, \{t_n\}]} S' \xrightarrow[\epsilon]{\Gamma} C_S [S'[X/t_1], \dots, S'[X/t_n]]$$

using **Ctnx**, **Flat**, **Dist** and **Bind**.

*Proof.* By the previous proposition we can reduce it to:

$$C_S [\bigcup_{X \in \{t_1\}} S', \dots, \bigcup_{X \in \{t_n\}} S']$$

and now, by **Ctnx** and **Bind** to:

$$C_S [S'[X/t_1], \dots, S'[X/t_n]]$$

We split the result of completeness in two simpler lemmas. The first one shows that any *SAS* obtained by *SRL* can be refined by the relation *SNarr*, in the sense that it can achieve a set-expression with more information.

**Lemma 4 (Completeness I).** *If  $S \triangleleft C$  then for any  $\Gamma$  with  $\Gamma \cap FV(S) = \emptyset$  we can derive  $S \xrightarrow[\epsilon]{\Gamma^*} S'$  such that  $C \sqsubseteq (S')^*$*

*Proof.* By induction on the length  $l$  of the proof for  $S \triangleleft C$ :

$l = 1$  In one step the possible proofs are  $S \triangleleft \{\perp\}$ ,  $\{X\} \triangleleft \{X\}$ ,  $\{c\} \triangleleft \{c\}$  with  $c \in DC^0 \cup \{\mathbb{F}\}$  or  $f(\bar{t}) \triangleleft \{\mathbb{F}\}$  where  $f(\bar{t})$  has a clash with every rule for  $f$  in the program. The first case is trivial; for the next two, as  $\{X\}$  and  $\{c\}$  are normal forms, in 0 derivation steps of *SNarr* we have the required  $S'$ ; for the last one by **Nrrw**<sub>2</sub> we can obtain the *SAS*  $\{\mathbb{F}\}$ .

$l > 1$  possible proofs are:

- by rule 3 the proof is  $\{c(\bar{t})\} \triangleleft \{c(\bar{t}')\}$ , where  $\bar{t}' \sqsubseteq \bar{t}$  by Prop. 2. With *SNarr*, in 0 steps with  $\theta = \epsilon$  we obtain  $\{c(\bar{t})\}$  as a *SAS* for  $\{c(\bar{t})\}$  and clearly  $\{c(\bar{t}')\} \sqsubseteq \{c(\bar{t})\}$
- by rule 4 we have:

$$\frac{S\sigma \triangleleft C}{f(\bar{t}) \triangleleft C}$$

where  $(f(\bar{s}) \rightarrow S) \in \mathcal{P}$  and  $\sigma \in CSubst_{\mathbb{F}}$  ( $\sigma$  does not need to introduce  $\perp$  because  $f(\bar{t})$  does not contain  $\perp$ ) such that  $f(\bar{s})\sigma = f(\bar{t})$ . Clearly  $\sigma \upharpoonright_{var(\bar{s})}$  is a m.g.u. for  $f(\bar{t})$  and  $f(\bar{s})$ , and  $Dom(\sigma) \cap \Gamma = \emptyset$ . Then, by **Nrrw**<sub>1</sub> we can derive

$$f(\bar{t}) \xrightarrow[\epsilon]{\Gamma} S\sigma$$

By i.h., as  $S\sigma \triangleleft C$  then  $S\sigma \xrightarrow[\epsilon]{\Gamma^*} S'$  where  $C \sqsubseteq (S')^*$ .

- by rule 6 we have:

$$\frac{S \triangleleft \{\mathbb{F}\}}{fails(S) \triangleleft \{true\}}$$

By i.h. we have  $S \xrightarrow[\epsilon]{\Gamma^*} S'$  such that  $\{\mathbb{F}\} \sqsubseteq (S')^*$ , what means  $(S')^* = \{\mathbb{F}\}$ . By

(possible several applications of) the rule **Cntx** we have  $fails(S) \xrightarrow[\epsilon]{\Gamma^*} fails(S')$ ,

and then by **Fail**<sub>1</sub> we obtain the *SAS*  $\{true\}$  as it was expected.

- by rule 7 we have:

$$\frac{S \triangleleft C}{fails(S) \triangleleft \{false\}}$$

having  $t \in C - (\{\perp, \mathbb{F}\})$ . By i.h. we can derive  $fails(S) \xrightarrow[\epsilon]{\Gamma^*} S'$  with  $C \sqsubseteq (S')^*$ .

Then it must exist some  $t' \in (S')^*$  with  $t \sqsubseteq t'$ , what in particular means  $t' \notin \{\perp, \mathbb{F}\}$ . Then by rule **Fail**<sub>2</sub> we have  $S' \xrightarrow[\epsilon]{\Gamma} \{false\}$  as expected.

– by rule 8 it must be  $\mathcal{S} = \bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2$  and the proof has the form:

$$\frac{\mathcal{S}_1 \triangleleft \mathcal{C}_1 \quad \mathcal{S}_2[X/\mathcal{C}_1] \triangleleft \mathcal{C}}{\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2 \triangleleft \mathcal{C}}$$

As  $\Gamma \cap FV(\mathcal{S}) = \emptyset$ , in particular  $\Gamma \cap FV(\mathcal{S}_1) = \emptyset$ , and we also have  $X \notin FV(\mathcal{S}_1)$ .

Then i.h., as  $\mathcal{S}_1 \triangleleft \mathcal{C}_1$  we can derive  $\mathcal{S}_1 \xrightarrow[\epsilon]{\Gamma \cup \{X\}} \mathcal{S}'_1$  such that  $\mathcal{C}_1 \sqsubseteq (\mathcal{S}'_1)^*$ . So, if

the p.c.f. of  $\mathcal{S}'_1$  is  $C_{\mathcal{S}'_1} [t_1, \dots, t_n]$  we have  $\mathcal{C}_1 \sqsubseteq \{t_1, \dots, t_n\}$ .

By the rule **Cntx** we can perform the derivation:

$$\bigcup_{X \in \mathcal{S}_1} \mathcal{S}_2 \xrightarrow[\epsilon]{\Gamma} \bigcup_{X \in C_{\mathcal{S}'_1} [t_1, \dots, t_n]} \mathcal{S}_2$$

and by Prop. 4 it can be reduced into:

$$C_{\mathcal{S}'_1} [\mathcal{S}_2[X/t_1], \dots, \mathcal{S}_2[X/t_n]]$$

Now, as  $\mathcal{S}_2[X/\mathcal{C}_1] \triangleleft \mathcal{C}$  and  $\mathcal{C}_1 \sqsubseteq \{t_1, \dots, t_n\}$ , by Prop. 4 and then Prop. 6 we have  $\mathcal{S}_2[X/\{t_1, \dots, t_n\}] \triangleleft \mathcal{C}'$  with  $\mathcal{C} \sqsubseteq \mathcal{C}'$ . This can be decomposed by rule 9' of *CRWL* into  $n$  proofs  $\mathcal{S}_2[X/\{t_i\}] \triangleleft \mathcal{C}'_i$  such that  $\mathcal{C}'_1 \cup \dots \cup \mathcal{C}'_n = \mathcal{C}'$ . By i.h. for each  $i \in \{1, \dots, n\}$  we have:

$$\mathcal{S}_2[X/t_i] \xrightarrow[\epsilon]{\Gamma} \mathcal{S}_2^i$$

such that  $\mathcal{C}'_i \sqsubseteq (\mathcal{S}_2^i)^*$ , i.e., if  $C_{\mathcal{S}_2^i} [\bar{s}_i]$  is the p.c.f. for  $\mathcal{S}_2^i$ , we have  $\mathcal{C}'_i \sqsubseteq \{\bar{s}_i \tau_i\}$  ( $\tau_i$  replaces the produced variables of  $C_{\mathcal{S}_2^i}$  by  $\perp$ ). By the rule **Cntx** for the whole expression we can mimic these derivation steps to obtain  $\mathcal{S}'$ :

$$C_{\mathcal{S}'_1} [\mathcal{S}_2[X/t_1], \dots, \mathcal{S}_2[X/t_n]] \xrightarrow[\epsilon]{\Gamma} C_{\mathcal{S}'_1} [C_{\mathcal{S}_2^1} [\bar{s}_1], \dots, C_{\mathcal{S}_2^n} [\bar{s}_n]] = \mathcal{S}'$$

So,  $\mathcal{S}'$  is a composition of principal contexts that, by Prop. 8, has a p.c.f. like:

$$C [\bar{s}_1, \dots, \bar{s}_n]$$

whose information set is  $(\mathcal{S}')^* = \{\bar{s}_1 \tau, \dots, \bar{s}_n \tau\}$ , where  $\tau$  coincides with the composition  $\tau = \tau_1 \dots \tau_n$ . As for each  $i$  we have  $\mathcal{C}'_i \sqsubseteq \{\bar{s}_i \tau_i\}$ , joining the sets we have then  $\mathcal{C}' \sqsubseteq \{\bar{s}_1 \tau, \dots, \bar{s}_n \tau\}$ . An finally, by transitivity of  $\sqsubseteq$ , as  $\mathcal{C} \sqsubseteq \mathcal{C}'$  we have  $\mathcal{C} \sqsubseteq \{\bar{s}_1 \tau, \dots, \bar{s}_n \tau\} = (\mathcal{S}')^*$ .

– by rule 9' it must be  $\mathcal{S} = \mathcal{S}_1 \cup \dots \cup \mathcal{S}_n$  and the proof has the form:

$$\frac{\mathcal{S}_1 \triangleleft \mathcal{C}_1 \quad \dots \quad \mathcal{S}_n \triangleleft \mathcal{C}_n}{\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n \triangleleft \mathcal{C}_1 \cup \dots \cup \mathcal{C}_n}$$

By i.h. for each  $i \in \{1, \dots, n\}$  we can derive  $\mathcal{S}_i \xrightarrow[\perp]{\Gamma} \mathcal{S}'_i$  such that  $\mathcal{C}_i \sqsubseteq (\mathcal{S}'_i)^*$ .

Then by rule **Cntx** we can derive:

$$\mathcal{S}_1 \cup \dots \cup \mathcal{S}_n \xrightarrow[\epsilon]{\Gamma} \mathcal{S}'_1 \cup \dots \cup \mathcal{S}'_n = \mathcal{S}'$$

The information set of such expression is  $(S')^* = (S'_1)^* \cup \dots \cup (S'_n)^*$  that clearly refines  $\mathcal{C} = \mathcal{C}_1 \cup \dots \cup \mathcal{C}_n$

For this result we see the relation  $SNarr$  as a pure rewriting relation: the answer substitution is  $\epsilon$ . The next lemma shows that any substitution  $\theta$  making reducible a set-expression  $\mathcal{S}$  is captured or generalized by a narrowing derivation from  $\mathcal{S}$ . For example, if we have the rule  $f(z) \rightarrow \{s(z)\}$ , it is clear that  $SNarr$  could derive  $f(z) \xrightarrow[\epsilon]{\Gamma} \{s(z)\}$ . What the next lemma proves is that in fact there is a narrowing

derivation  $f(X) \xrightarrow[\frac{\Gamma}{[X/z]}]{\epsilon} \{s(z)\}$ , that is, the value  $z$  for the variable  $X$  of the set-expression  $f(X)$  can be found.

**Lemma 5 (Completeness II).** *Let  $\mathcal{S} \in SetExp$ ,  $\theta \in CSubst$  and  $\Gamma$  such that  $\Gamma \cap PV(\mathcal{S}) = \emptyset$ . If  $\mathcal{S}\theta \xrightarrow[\epsilon]{\Gamma} \mathcal{S}'$  then there exists  $\theta', \mu \in CSubst$ , with  $\theta = \theta'\mu$ , such that  $\mathcal{S} \xrightarrow[\theta']{\Gamma} \mathcal{S}''$  with  $\mathcal{S}' = \mathcal{S}''\mu$ .*

*Proof.* It is enough to prove the result for one derivation step (the extension to  $n$  steps is direct by induction on  $n$ ). Moreover, in the case of rule **Cntx** we assume that the sub-derivation is performed by another rule of  $SNarr$  distinct from **Cntx**. We reason for each rule of  $SNarr$ :

**Cntx** In this case we assume a derivation of the form  $C\theta[\mathcal{S}]\theta \xrightarrow[\epsilon]{\Gamma} C\theta[\mathcal{S}']$  and we

must prove that  $C[\mathcal{S}] \xrightarrow[\theta']{\Gamma} C\theta'[\mathcal{S}']$ , where  $\theta = \theta'\mu$  and  $C\theta[\mathcal{S}'] = (C\theta'[\mathcal{S}''])\mu$ .

The first derivation uses the sub-derivation  $\mathcal{S}\theta \xrightarrow[\epsilon]{\Gamma \cup PV(C)} \mathcal{S}'$  performed by some rule of  $SNarr$  distinct from **Cntx**. Then, for such sub-derivation (as we prove next) we have  $\mathcal{S} \xrightarrow[\theta']{\Gamma \cup PV(C)} \mathcal{S}''$ , with  $\theta = \theta'\mu$  and  $\mathcal{S}' = \mathcal{S}''\mu$ . Then, by **Cntx**

we have  $C[\mathcal{S}] \xrightarrow[\theta']{\Gamma} C\theta'[\mathcal{S}']$ . Now, we have  $(C\theta'[\mathcal{S}''])\mu = C\theta'\mu[\mathcal{S}''\mu]$ , and as  $\theta'\mu = \theta$  and  $\mathcal{S}''\mu = \mathcal{S}'$  we have  $(C\theta'[\mathcal{S}''])\mu = C\theta[\mathcal{S}']$ , as we expect.

**Nrrw<sub>1</sub>** Assume  $f(\bar{t})\theta \xrightarrow[\epsilon]{\Gamma} \mathcal{S}\sigma$ , where  $(f(\bar{s}) \rightarrow \mathcal{S}) \in \mathcal{P}$  and  $\sigma$  is a m.g.u. for  $\bar{s}$  and  $\bar{t}\theta$ , with  $Dom(\sigma) \cap \Gamma = \emptyset$ . Then we have  $\bar{s}\sigma = \bar{t}\theta\sigma$ . Moreover, as the derivation step proceeds with an empty substitution,  $\sigma$  cannot affect to the variables of  $\bar{t}\theta$ , so we have  $\bar{t}\theta = \bar{s}\sigma$  and  $Dom(\sigma) = var(\bar{s})$ . As  $\bar{t}$  and  $\bar{s}$  are unifiable, there exist a m.g.u.  $\theta'$  for them, and it must satisfy  $\theta' \sqsubseteq \theta$ ,  $\theta' \sqsubseteq \sigma$ . Then we can derive:

$$f(\bar{t}) \xrightarrow[\theta'|_{var(\bar{t})}]{\Gamma} \mathcal{S}\theta' = \mathcal{S}''$$

Now we must look for the appropriate  $\mu$ . As the instance of the program rule uses fresh variables and  $Dom(\sigma) = var(\bar{s})$ , we have  $Dom(\theta) \cap Dom(\sigma) = \emptyset$ . In this situation we can get  $\mu \in CSubst$  such that simultaneously satisfy  $\theta'\mu = \theta|_{var(\bar{t})}$

and  $\theta'\mu = \sigma$ . And then we have the following equivalences  $\mathcal{S}''\mu = \mathcal{S}\theta'\mu = \mathcal{S}\sigma$ , what is what we need.

**Nrrw<sub>2</sub>** Assume  $f(\bar{t})\theta \xrightarrow[\square]{\Gamma} \{\mathbb{F}\}$ , where  $f(\bar{t})\theta$  has a  $DC \cup \{\mathbb{F}\}$ -clash with all the heads of the rules of the program. As our program is *CIS* this conflict must be because  $\bar{t}\theta$  has  $\mathbb{F}$  at a position where the heads has a constructor of *DC*. The substitution  $\theta$  cannot introduce this  $\mathbb{F}$  in  $f(\bar{t})\theta$ , but it was previously in  $f(\bar{t})$ . Then  $f(\bar{t})$  has exactly the same clash, so we can derive:

$$f(\bar{t}) \xrightarrow[\epsilon]{\Gamma} \{\mathbb{F}\}$$

In this case  $\theta' = \epsilon$  and  $\mu = \theta$ .

**Fail<sub>1</sub>** Assume  $\text{fails}(\mathcal{S})\theta \xrightarrow[\epsilon]{\Gamma} \{\text{true}\}$ , where  $(\mathcal{S}\theta)^* = \{\mathbb{F}\}$ . As  $\theta \in \text{CSubst}$  it must be  $\mathcal{S}^* = \{\mathbb{F}\}$ , and then, clearly  $\text{fails}(\mathcal{S}) \xrightarrow[\epsilon]{\Gamma} \{\text{true}\}$ , with  $\theta' = \epsilon$  and  $\mu = \theta$ .

**Fail<sub>2</sub>** Assume  $\text{fails}(\mathcal{S})\theta \xrightarrow[\epsilon]{\Gamma} \{\text{false}\}$ , such that there exist  $t \in (\mathcal{S}\theta)^* - \{\perp, \mathbb{F}\}$ . Then there must be a corresponding  $t' \in \mathcal{S}^*$  such that  $t' = t\theta$ , so  $t' \notin \{\perp, \mathbb{F}\}$ . Clearly  $\text{fails}(\mathcal{S}) \xrightarrow[\epsilon]{\Gamma} \{\text{false}\}$ , with  $\theta' = \epsilon$  and  $\mu = \theta$ .

**Flat** This case is clear taking  $\theta' = \epsilon$  and  $\mu = \theta$ .

**Dist** Idem.

**Bind** If  $\bigcup_{X \in \{t\}} \mathcal{S}\theta \xrightarrow[\square]{\Gamma} \mathcal{S}\theta[X/t\theta]$  then clearly we can also derive  $\bigcup_{X \in \{t\}} \mathcal{S} \xrightarrow[\epsilon]{\Gamma} \mathcal{S}[X/t]$ . In this case  $\theta' = []$  and  $\mu = \theta$ .

**Elim** Assume  $\bigcup_{X \in \mathcal{S}'\theta} \mathcal{S}\theta \xrightarrow[\square]{\Gamma} \mathcal{S}\theta$ , with  $X \notin \text{FV}(\mathcal{S})$ . Then, as  $\theta$  does not affect to  $X$ , this variable cannot occur in  $\mathcal{S}$ , so we can also derive  $\bigcup_{X \in \mathcal{S}'} \mathcal{S} \xrightarrow[\epsilon]{\Gamma} \mathcal{S}$ . In this case  $\theta' = []$  and  $\mu = \theta$ .

Joining these two lemmas we obtain our completeness result.

**Theorem 3 (Completeness).** *If  $\vdash_{\text{SRL}} \mathcal{S}\theta \triangleleft \mathcal{C}$  and  $\Gamma \cap \text{FV}(\mathcal{S}) = \emptyset$ , then  $\theta = \theta'\mu$  and  $\mathcal{S} \xrightarrow[\theta']{\Gamma}^* \mathcal{S}'$  such that  $\mathcal{C} \sqsubseteq (\mathcal{S}'\mu)^*$ .*

*Proof.* If  $\mathcal{S}\theta \triangleleft \mathcal{C}$ , by Lemma 4  $\mathcal{S}\theta \xrightarrow[\epsilon]{\Gamma}^* \mathcal{S}''$  such that  $\mathcal{C} \sqsubseteq (\mathcal{S}'')^*$ . Then, by Lemma 5,  $\theta$  can be decomposed as  $\theta = \theta'\mu$  and  $\mathcal{S} \xrightarrow[\theta']{\Gamma}^* \mathcal{S}'$ , such that  $\mathcal{S}'' = \mathcal{S}'\mu$ . Joining the results, we have  $\mathcal{C} \sqsubseteq (\mathcal{S}'\mu)^*$

As an easy consequence of this theorem we obtain:

**Corollary 5.** *If  $\mathcal{P} \vdash_{\text{SRL}} \mathcal{S}\theta \triangleleft \{\mathbb{F}\}$ , then  $\mathcal{S} \xrightarrow[\theta']{\emptyset}^* \{\mathbb{F}\}$  for some  $\theta'$  such that  $\theta \sqsubseteq \theta'$ .*

This result shows that we have achieved our goal of devising an operational mechanism for failure, which is *constructive*, in the sense that in presence of variables is able to find appropriate substitutions for them.

## 5 Conclusions and Future Work

We have defined a narrowing relation for functional logic programs which can make use of failure as programming construct, by means of a function  $fails(e)$ , which is a computable approximation to the property ‘ $e$  cannot be reduced to head normal form’. Programs are written in a set-oriented syntax where expressions can use unions and indexed unions. Thus the syntax directly reflects non-determinism of functions and call-time choice semantics in a suitable way as to facilitate the definition of narrowing of set-expressions.

The set-narrowing relation serves to the purpose of computing failures, since the failure of an expression  $e$  corresponds to the fact that the expression can be narrowed to the set  $\{F\}$ , where  $F$  is a constant introduced to represent failure. An important feature of our notion of narrowing is that it can operate to compute failures even in presence of variables, for which appropriate bindings are obtained. Using a frequent terminology in the context of logic programming and negation, our narrowing relation realizes *constructive failure* for functional logic programs. Nothing similar can be found in existing *FLP* systems like *Curry* or *TOY*.

The definition of set-narrowing is quite amenable for implementation, and we have indeed implemented a first prototype for it, with which the examples in the paper have been executed.

In this paper we have not addressed the issue of *strategies* for set-narrowing. Nevertheless, for the implementation we have used some kind of ‘demand driven strategy’ close to the usual one in existing systems. In this sense it is interesting to remark that, in contrast to other notions of narrowing proposed for non-determinism with call-time choice [7, 8], which has been criticized [3] as being too far from real computations, our definition of set-narrowing seems better suited to the adoption of strategies. Notice that, in particular, the rule  $\mathbf{Nrrw}_1$  performs exactly a narrowing step in the classical sense, just with some conditions imposed about what variables can be bound.

The theoretical investigation of set-narrowing strategies, as well as the integration of our implementation in the system *TOY* are the main subjects of future work.

## References

1. M. Abengózar-Carneros et al. *TOY* a multiparadigm declarative language. version 2.0. Technical report, Depto. SIP, UCM Madrid, January 2001.
2. S. Antoy. Definitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming*, pages 143–157. Springer LNCS 632, 1992.
3. Antoy S. Constructor-based conditional narrowing. In *PPDP'01*. Springer LNCS, 2001.
4. K.R. Apt and R. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19&20:9–71, 1994.
5. D. Chan. Constructive negation based on the completed database. In *Proc. 5th ICSLP*, pages 111–125, 1988.

6. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
7. J.C. González-Moreno, T. Hortalá-González, F.J. López-Fraguas, and M. Rodríguez-Artalejo. A rewriting logic for declarative programming. In *Proc. European Symposium on Programming (ESOP'96)*, pages 156–172. Springer LNCS 1058, 1996.
8. J.C. González-Moreno, T. Hortalá-González, F.J. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *Journal of Logic Programming*, 40(1):47–87, 1999.
9. M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
10. M. Hanus (ed.). Curry: An integrated functional logic language. Available at <http://www-i2.informatik.rwth-aachen.de/~hanus/curry/report.html>, February 2000.
11. F.J. López-Fraguas and J. Sánchez-Hernández. *TOY*: A multiparadigm declarative system. In *Rewriting Techniques and Applications, RTA'99*, volume 1631 of *Springer LNCS*, pages 244–247, 1999.
12. F.J. López-Fraguas and J. Sánchez-Hernández. Proving failure in functional logic programs. In *Proc. CL'00*, number 1861 in Springer LNAI, pages 179–193, 2000.
13. F.J. López-Fraguas and J. Sánchez-Hernández. Functional logic programming with failure: A set-oriented view. In *Proc. LPAR'01*, number 2250 in Springer LNAI, pages 455–469, 2001.
14. F.J. López-Fraguas and J. Sánchez-Hernández. A proof theoretic approach to failure in functional logic programming. Available at <http://www.ucm.es/info/dsip/jaime/tplp.ps>, 2002.
15. J.J. Moreno-Navarro. Default rules: An extension of constructive negation for narrowing-based languages. In *Proc. Eleventh International Conference on Logic Programming*, pages 535–549. MIT Press, 1994.
16. J.J. Moreno-Navarro. Extending constructive negation for partial functions in lazy functional-logic languages. In *Proc. 5th International Workshop on Extensions of Logic Programming*, pages 213–227. Springer LNAI 1050, 1996.
17. J.C. Reynolds. *Theories of Programming Languages*, Cambridge University Press, 1988
18. P.J. Stuckey. Constructive negation for constraint logic programming. In *Proc. Logic in Computer Science (LICS'91)*, pages 328–339, 1991.
19. P.J. Stuckey. Negation and constraint logic programming. *Information and Computation*, 118:12–33, 1995.