**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

**Research
Report**

RR-92-54

A Direct
Semantic Characterization of RELFUN

Harold Boley

November 1992

**Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH**

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

Intelligent Engineering Systems
Intelligent User Interfaces
Computer Linguistics
Programming Systems
Deduction and Multiagent Systems
Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

A Direct Semantic Characterization of RELFUN*

Harold Boley

DFKI-RR-92-54

To appear in:
Proc. 3rd International Workshop on Extensions of Logic Programming. ELP'92,
DEIS, Univ. Bologna, Italy, February 1992,
Springer, Lecture Notes in Artificial Intelligence, LNAI 660, 1993.

This work has been supported ba a grant from The Federal Ministry for Research and Technology (FKZ ITWM-8902 C4).

A Direct Semantic Characterization of RELFUN[†]

Harold Boley
Deutsches **F**orschungszentrum für **K**ünstliche **I**ntelligenz
Box 2080, D-6750 Kaiserslautern, F. R. Germany
boley@informatik.uni-kl.de

Abstract

This paper attempts a direct semantic formalization of first-order relational-functional languages (the characteristic RELFUN subset) in terms of a generalized model concept. Function-defining conditional equations (or, footed clauses) and active call-by-value expressions (in clause premises) are integrated into first-order theories. Herbrand models are accomodated to relational-functional programs by not only containing ground atoms but also ground molecules, i.e. specific function applications paired with values. Extending SLD-resolution toward innermost conditional narrowing of relational-functional clauses, SLV-resolution is introduced, which, e.g., flattens active expressions. The T_P-operator is generalized analogously, e.g. by unnesting ground-clause premises. Soundness and completeness proofs for SLV-resolution naturally extend the corresponding results in logic programming.

1 Introduction

RELFUN is a logic language primarily extended by call-by-value (eager) functions that may be non-ground, non-deterministic, varying-arity, and higher-order. These functions are defined by extended Horn clauses having a 'foot' premise for value returning. This extension can also be viewed as (directed) conditional equations permitting 'extra' variables in conditions, which may accumulate partial results. It entails the following syntactic changes of PROLOG:

Footed clauses: Starting with DATALOG, "`:-`"-rules **may be augmented** by an ampersand infix, "`&`", between the normal body premises and the foot premise; facts (empty bodies), by a joined infix, "`:-&`".

Active expressions: Proceeding to PROLOG, passive structures **are rewritten** using square brackets, "`[...]`", reserving round parentheses, "`(...)`", for RELFUN's active call-by-value expressions (permitted in premises).

*This research was supported by the BMFT under Grant ITW 8902 C4.
†This research was supported by the BMFT under Grant ITW 8902 C4.

As shown by the Fibonacci programs in example 1, RELFUN's function-defining footed clauses (e.g. for `fibfun`) can be developed from PROLOG-like relation-defining Horn clauses (e.g. for `fibrel`) via an intermediate footed-clause form using a generalized relational `is`-primitive in a functional, `let`-like manner (e.g. for `fibfis`). When reading such clauses we extend PROLOG's "*... if ...*" for "`... :- ...`" to "*... if ... returns ...*" for "`... :- ... & ...`" (or just "*... returns ...*" for "`... :-& ...`"). [1]

Example 1 *Recursive Fibonacci relations and functions in RELFUN.*

```
fibris(N,F) :- F is fibfun(N).

fibrel(0,s[0]).
fibrel(s[0],s[0]).
fibrel(s[s[N]],F) :-  fibrel(N,X), fibrel(s[N],Y), plusrel(X,Y,F).

fibfis(0)       :-&  s[0].
fibfis(s[0])    :-&  s[0].
fibfis(s[s[N]]) :-   X is fibfis(N), Y is fibfis(s[N]) & plusfun(X,Y).

fibfun(0)       :-&  s[0].
fibfun(s[0])    :-&  s[0].
fibfun(s[s[N]]) :-&  plusfun(fibfun(N),fibfun(s[N])).


plusrel(0,N,N).
plusrel(s[M],N,P) :-  plusrel(M,s[N],P).

plusfun(0,N)    :-&  N.
plusfun(s[M],N) :-&  plusfun(M,s[N]).
```

Relation definitions in RELFUN employ generalized Horn clauses, namely 'hornish' clauses, which may again call arbitrary functions, either within any argument of a relation call or the right-hand side (rhs) of the `is`-primitive (e.g. in `fibris`). So the body premises of hornish clauses are relational on the **top-level** (just binding variables, like Horn-clause premises), but may **contain** functional applications (also returning values). Conversely, the head and foot of footed clauses can be regarded as the two sides of an equation, giving these clauses a principal functional flavor, although their body conditions are exactly like the relational top-level premises of hornish clauses. Altogether, RELFUN's clauses tightly integrate relational and functional characteristics. [2]

[1] While the infix "`:-&`" corresponds to a (directed, unconditional) "$=$", the mixfix "`:- ... &`" corresponds to a (directed, conditional) "$\stackrel{...}{=}$". However, we will not formalize functions using a logic with a distinguished (directed) equality **predicate**, but will 'build in' "`:-&`" and "`:- ... &`" even more deeply, as new **connectives**.

[2] Still, rather than indiscriminately speaking of 'relational-functional' language constructs, we will didactically distinguish 'relational' and 'functional' constructs on the basis of their principal characteristics.

The following functional version of J. W. Lloyd's relational `slowsort` example [7] shows the use of non-ground and non-deterministic subfunction calls for defining a deterministic main function.

Example 2 *A functional slowsort program in RELFUN.*

```
% Sort filters non-deterministic permutations through sorted:
sort(X) :-& sorted(perm(X)).

% Return sorted lists unchanged, fail for unsorted ones:
sorted([]) :-& [].
sorted([X]) :-& [X].
sorted([X,Y|Z]) :- lesseq(X,Y) & cons(X,sorted([Y|Z])).

% Permute by a non-ground delete call returning U-less lists
% and binding U for a cons call enclosing the perm recursion:
perm([]) :-& [].
perm([X|Y]) :-& cons(U,perm(delete(U,[X|Y]))).

% Non-deterministically delete X elements from list argument:
delete(X,[X|Y]) :-& Y.
delete(X,[Y|Z]) :-& cons(Y,delete(X,Z)).

% A less-or-equal relation over s-terms:
lesseq(0,X).
lesseq(s[X],s[Y]) :- lesseq(X,Y).

% cons(h,t) calls h and t by value, [h|t] only instantiates h and t:
cons(X,Y) :-& [X|Y].
```

Since programs for Fibonacci numbers, list sorting, and many other purposes are normally used in a deterministic mode, we think they should be formulated as functions rather than relations, indicating the preferred direction of computation. However, in RELFUN such functions still permit inverse calls (e.g. `s[0] is fibfun(W)` non-deterministically binds `W` to `0` or `s[0]`) and can make natural internal use of relations (e.g. `lesseq`) and non-deterministic functions (e.g. `perm` and `delete`).

A comprehensive overview of RELFUN and related work as well as pointers to its applications and to its original operational (interpretative), LISP-implemented semantics can be found in [3]. Among the tools of the RELFUN implementation there is a term-rewriting algorithm `relationalize` for transforming footed and hornish clauses into Horn clauses, thus indirectly characterizing their model-theoretic semantics. However, this semantic indirectness makes our understanding of functions totally dependent on our understanding of relations (inverting the dependency incurred by the LISP-based interpreter), whereas we work towards "equal declarative depth" for both of them.

The present paper thus attempts to directly characterize the semantics of 'basic' RELFUN, the pure-RELFUN subset exemplifying fixed-arity first-order relational-functional languages, in terms of a generalized model concept: RELFUN models contain both **atoms** (relations) and **directed unconditional equations** (functions). This would permit a common foundation of logic and functional programming, reducing the gap between these declarative paradigms. Through a model-theoretic foundation of relational-functional languages, the semantic characteristics available or lacking in either of these declarative-programming paradigms can be assessed in a way more neutral than via the indirection of mutual implementions of, and cross-translations between these paradigms. For instance, on the basis of our characterization we can study such questions as "How will functional call-by-value expressions enrich (and complicate) the semantics of relational languages?" or "How will the relational meaning of non-ground arguments carry over to the functional meaning of arguments and returned values?" Another important motivation of the present work is to make the many alternative relational-functional integration proposals (see, e.g., [1] and [4]) comparable on a common ground, revealing their deeper, non-syntactic differences. Finally, we think the model-theoretic treatment can provide us with a long-term yardstick for developing a 'minimal' integration of the essential concepts of relational and functional languages: in the multitude of integration proposals, only "Occam's razor" can help sorting out the **proper integration constructs** from other "nice features".

In fact, with basic RELFUN we have attempted to operationally explore a tight, minimum integration of the concepts of a relation and a function themselves. Among other things, the classical eager functional expressions (innermost reduction) have been extended to non-deterministic function nestings to accomodate relational non-determinism. Then, the semantic interpretation of functions just uses mappings to **sets** of domain individuals, and expressions are semantically evaluated using **expression assignments**, a natural, set-valued extension of relational term assignments. These semantic extensions are less complicated than the semantics of lazy expressions (outermost reduction) as a relational-functional integration concept, as introduced by other recent proposals (e.g., K-LEAF [6] and BABEL [8]): eagerness keeps the semantics strict and simple, whereas laziness accepts the non-strictness overhead to give a meaning to unifications involving non-terminating expressions. While basic RELFUN's operational integration concepts may be close to a minimum, its current model-theoretic characterization is still quite preliminary and will certainly need further simplification and improvement.

On the other hand, pure-RELFUN extensions of the present treatment could directly incorporate the semantics of varying-arity operations, which can also be reduced to unary ones over lists. Similarly, RELFUN's higher-order operations should not be too difficult to add, as they are restricted to those reducible to first-order operations using an `apply` dummy as introduced for corresponding PROLOG extensions by D. H. D. Warren [10]. While these two extensions have long existed in the implemented RELFUN system, further extensions such as finite domains will first require their own operational test phase before we can think of including them in the formal semantics. Finally, some aspects of our RELFUN extensions of SLD-resolution, Herbrand models, and T_P-operators will probably be transferable to other languages.

Our basic semantic treatment draws heavily on chapters 1 and 2 of J. W. Lloyd's book [7], construing a parallel between first-order relations and first-order functions, enabled by suitably generalizing the latter in a non-ground, non-deterministic fashion. This relational-functional parallel in the formal definitions given here derived from considerations in language design such as expressive power, orthogonality, and uniformity of constructs. But it also simplifies transferring foundation theorems of logic programming (as found, e.g., in J. W. Lloyd's book) to eager, non-ground, non-deterministic first-order functional programming and to unified relational-functional programming. We think that a fundament for functional programming should be 'grounded' on a level as deep as the (Herbrand-)model-theoretic fundament of relational programming. Specifically this means that we will try to establish function definitions as subsets $\{f(a_1, ..., a_n)$ `:-&` $b,\ \ldots\}$ of so-called ground 'molecules' (directed unconditional equations) from the Herbrand 'cross' just like relation definitions are established as subsets $\{r(a_1, ..., a_n),\ \ldots\}$ of ground atoms from the Herbrand base. Intuitively, Herbrand cross models employ molecules for the 'pointwise' definition of a (discrete) function, akin to the familiar notion of the 'graph' (or 'extension') of a function as a set of pairs. Avoiding dependencies between the molecules of such a model which correspond to the usual 'functionality' restriction $f(a_1, ..., a_n)$ `:-&` $b \wedge f(a_1, ..., a_n)$ `:-&` $c \Longrightarrow b = c$, it will **simplify** this semantics that we permit $b \neq c$ i.e., non-deterministic functions. [3] Unaffected by non-determinism, the **directedness** of functional computation is expressed by the '$f(a_1, ..., a_n)$-to-b' order of each molecule $f(a_1, ..., a_n)$ `:-&` b in an Herbrand cross model.

On the basis of the unified pure-RELFUN constructs, the impure relational-functional features can also be introduced in a uniform manner. For instance, after proving results corresponding to the "independence of the computation rule" in [7], we could proceed from '`and`-parallel' to '`and`-sequential' relational-functional premise evaluation, which is the operational semantics actually implemented for RELFUN (just as for PROLOG). Similarly, the resolution/model-theoretic '`or`-parallelism' of relational-functional clauses could be weakened toward the operational (but implementation-incomplete!) '`or`-sequentialism' of backtracking. Finally, functions **and** relations can be forced to operate (more) deterministically using the same `cut`, `commit`, or substitute constructs; however, adapting our model-theoretic approach to such optional determinism specifications may be difficult because of the semantic problems with `cut`-like notions.

2 Extending First-Order Theories to First-Order Relational-Functional Theories

We now begin with the formal development of first-order relational-functional programming by 'functionally' extending the "Foundations of Logic Programming" [7], which should also be consulted for references to classical work.

[3] (Re)specializing RELFUN to a sublanguage with only deterministic functions would cause semantic changes starting off from the interpretation concept. (While our non-deterministic function symbols are assigned mappings to the powerset of the domain, deterministic function symbols could be assigned constructor-like mappings to the domain itself.) Within models the 'deterministic-function' restriction could then be introduced as an axiom, but this would change Herbrand's **sets** to (non-free) **algebras**.

A *first-order relational-functional theory* consists of:

1. An alphabet.
2. A first-order relational-functional language (the well-formed formulas of the theory).
3. A set of axioms (a designated subset of the well-formed formulas).
4. A set of inference rules.

Definition 1 *The* alphabet *of a first-order relational-functional theory consists of nine classes of symbols (some notational conventions are given in parentheses, where all letters used may be subscripted):*

1. *Variables (normally denoted by the letters x, y, and z).* [4]
2. *Constants (normally denoted by the letters a, b, and c).*
3. *Constructors*[5] *(normally denoted by the letters j, k, and l).*
4. *Function symbols (normally denoted by the letters f, g, and h).*
5. *Relation symbols*[6] *(normally denoted by the letters p, q, and r).*
6. *Functional connectives (two binary infixes denoted by* `is` *and* `:-&` *and a ternary mixfix denoted by* `:-` *together with* `&`*).*
7. *Relational connectives (a unary prefix denoted by* $\neg$ *and binary infixes denoted by* $\wedge$, $\vee$, `:-`, *and* $\leftrightarrow$*).*[7]
8. *Quantifiers (denoted by* $\exists$ *and* $\forall$*).*
9. *Punctuation symbols (“*`[`*”, “*`]`*”, “*`(`*”, “*`)`*”, and “*`,`*”).*

The union of the classes of function and relation symbols will be referred to as operation symbols *or, briefly,* operators.

Note that RELFUN's implemented operational semantics does not differentiate subclasses for constructor, function, and relation symbols but contextually distinguishes **uses** of symbols from a united class, even permitting a given symbol to have occurrences in more than one subclass (e.g., the main operator symbol of a body premise will act as a relation but may re-occur in a foot premise, where it will act as a function; also, meta-calls make operators from constructors).

[4] In larger examples we will capitalize variable names and use digit suffixes instead of subscripts, e.g. x_1 becoming $X1$, to conform to RELFUN's actual PROLOG-like naming conventions.

[5] Often called “functors” or even “function symbols” in the literature.

[6] Often called “predicate symbols” in the literature.

[7] Much like in PROLOG's program clauses, “`:-`” without a consecutive “`&`” plays the role of “$\leftarrow$”.

Definition 2 *A* term *is defined inductively:*

1. *A variable is a term.*
2. *A constant is a term.*
3. *If k is an n-ary constructor and $t_1, \ldots, t_n$ are terms, then $k[t_1, \ldots, t_n]$ is a term, called a* structure.

The above use of square brackets for applying a constructor to arguments clearly sets off 'passive' structures from 'active' operator applications as defined below with the more usual round parentheses. In our semantic treatment of relational-functional languages the bracketing type serves readability but provides no information beyond that already implicit in the symbol classes, 'constructor' vs. 'operator'. In the implemented version of RELFUN, not distinguishing symbol classes, this information is exclusively conveyed by "[...]" vs. "(...)".

In RELFUN *cns* is employed as the binary list constructor (LISP's *cons* or "."), and *nil*, as usual, as the constant denoting the empty list. Externally, a *list term* having the right-recursively nested form $cns[t_1, cns[t_2, cns[\ldots, cns[t_n, t]\ldots]]]$ is written (PROLOG-like) as the linearized varying-arity term $[t_1, t_2, \ldots, t_n]$ for $t = nil$ or, $[t_1, t_2, \ldots, t_n|t]$ for t being a variable. However, we regard the varying-arity form as (passive) applications of a constructor *tup*, understood to precede unprefixed "[...]"-terms.

Definition 3 *An* expression *is defined inductively:*

1. *A term is an expression.*
2. *If f is an n-ary function symbol and $E_1, \ldots, E_n$ are expressions, then $f(E_1, \ldots, E_n)$ is an expression, called an* application*; if all of $E_1, \ldots, E_n$ are terms, $f(E_1, \ldots, E_n)$ is called a* flat application.

Such a notion of expressions is essential in functional programming, but lacks in non-extended logic programming (in [7], "expression" is given a different, peripheral meaning).

Definition 4 *A* (well-formed) formula *is defined inductively:*

1. *If r is an n-ary relation symbol and $E_1, \ldots, E_n$ are expressions, then $r(E_1, \ldots, E_n)$ is a formula, called a* relationship*; if all of $E_1, \ldots, E_n$ are terms, $r(E_1, \ldots, E_n)$ is called a* flat relationship *or, since this is the most basic kind of formula, an* atomic formula *or, simply, an* atom.
2. *If E is an expression and t is a term, then (t `is` E) is a formula, called a* setting formula *or, simply, a* setter*; if E is a flat application, (t `is` E) is called a* flat setter*; if E is a term, (t `is` E) is called a* term setter.

3. *If e is a flat application and E is an expression, then $(e$ `:-&` $E)$ is a formula; if E is a term, $(e$ `:-&` $E)$ is called a* molecular formula *or, simply, a* molecule.

4. *If e is a flat application, E is an expression, and W is a formula, then $(e$ `:-` W `&` $E)$ is a formula.*

5. *If W_1 and W_2 are formulas, then so are $(\neg W_1)$, $(W_1 \wedge W_2)$, $(W_1 \vee W_2)$, $(W_1$ `:-` $W_2)$, and $(W_1 \leftrightarrow W_2)$.*

6. *If W is a formula and x is a variable, then $(\exists x W)$ and $(\forall x W)$ are formulas.*

The restriction of e being a flat application in items *3.* and *4.* reflects the "constructor discipline" [9] of RELFUN's footed clauses. It could be dropped in a more general equational treatment of first-order relational-functional languages. Conversely, instead of letting W_1 be an arbitrary formula in $(W_1$ `:-` $W_2)$ of item *5.*, it could be immediately restricted to an atomic formula (flat relationship), as required for RELFUN's hornish clauses.

Note that the parentheses employed to build applications and relationships are indispensible parts of the syntax. The parentheses around entire formulas, however, are just used for grouping and will frequently be omitted if no ambiguities arise under the following partial precedence order: "$\neg$", "$\forall$", "$\exists$" precede "`is`" precedes "$\wedge$" precedes "$\vee$" precedes "`:-&`", "`:- ... &`", "`:-`", "$\leftrightarrow$".

There is a close kinship between flat setters and molecules, which will be confirmed in definition 16. Thus, an operation that switches between both formula types will be convenient.

Definition 5 *The self-inverse setter/molecule swapping operation "$^{\otimes}$" is defined as an exponentiation operator over sets of molecules, flat setters, and relationships (the u_i must be terms):*[8]

$$\begin{aligned} r(u_1,\ldots,u_m)^{\otimes} &= r(u_1,\ldots,u_m) \\ (t \texttt{ is } g(u_1,\ldots,u_m))^{\otimes} &= g(u_1,\ldots,u_m) \texttt{ :-\&} t \\ (g(u_1,\ldots,u_m) \texttt{ :-\&} t)^{\otimes} &= t \texttt{ is } g(u_1,\ldots,u_m) \\ \{F_1,\ldots,F_n\}^{\otimes} &= \{F_1^{\otimes},\ldots,F_n^{\otimes}\} \end{aligned}$$

Example 3 *a, b, c, x, y, $k[a,x,b]$, $l[y,y]$, and $k[a,l[y,y],b]$ are terms; $f(y,k[a,l[y,y],b],c,l[y,y])$ is a flat application; $r(b,f(y,k[a,l[y,y],b],c,l[y,y]))$ is a (non-flat) relationship. $f(y,k[a,l[y,y],b],c,l[y,y])$ `:-&` $k[a,x,b]$ is a molecule; $(f(y,k[a,l[y,y],b],c,l[y,y])$ `:-&` $k[a,x,b])^{\otimes} = k[a,x,b]$ `is` $f(y,k[a,l[y,y],b],c,l[y,y])$ is a flat setter.*

Definition 6 *The* first-order relational-functional language *given by an alphabet consists of the set of all formulas built from the symbols of the alphabet.*

[8] If "$^{\otimes}$" is applicable to a formula F, then $(F^{\otimes})^{\otimes} = F$.

In the following we will focus special kinds of formulas, namely RELFUN's clauses. Unaffected by their Horn-clause extensions (expressions, setters, and foot premises), they are closed formulas by assuming all variables to have a prenex universal quantifier.

Definition 7 *A* (program) clause *is a hornish (program) clause or a footed (program) clause. If w is an atomic formula, e is a flat application, $V_1, \ldots, V_n$ are relationships or setters, and E is an expression, then w :- $V_1, \ldots, V_n$, abbreviating w :- $(V_1 \wedge \ldots \wedge V_n)$, is a* hornish (program) clause *and e :- $V_1, \ldots, V_n$ & E, abbreviating e :- $(V_1 \wedge \ldots \wedge V_n)$ & E, is a* footed (program) clause. *w or e is the* head, *$V_1, \ldots, V_n$ is the* body, *and E is the* foot *of the clause. If $V_1, \ldots, V_n$ are all atoms, the hornish (program) clause w :- $V_1, \ldots, V_n$ is also called a* Horn (program) clause. *For $n = 0$, i.e. with an empty body, a hornish (program) clause w :- , abbreviating w :- true, is written as w, while a footed (program) clause e :- & E, abbreviating e :- true & E, is written as e :-& E.*

Definition 8 *A* ((first-order) relational-functional) program *P is a finite set of program clauses $\{c_1, \ldots, c_n\}$. P is usually written (with "."-terminators) as:*
c_1.
...
c_n.

A program will play the role of the set of axioms of a first-order relational-functional theory.

Definition 9 *The* empty (hornish) clause, *denoted , is the hornish clause of the form :- , which abbreviates false :- true. A* terminal ((t-)footed) clause, *denoted $\triangle(t)$, t a term, is a footed clause of the form :-& t, which abbreviates & t. The* trivial (hornish) clause, *denoted $\top$, is the hornish clause of the form true :- true.*

Definition 10 *A* relational goal *is a hornish clause of the form*

:- $V_1, \ldots, V_n$

that is, it has an empty head. A functional goal *is a footed clause of the form*

:- $V_1, \ldots, V_n$ & E

that is, it has an empty head.

It should be kept in mind that a relational goal is 'relational' in the usual sense only on the top-level: the V_i's need not be atoms but may be nested relationships or setters. Conversely, a functional goal may of course contain V_i's that are atoms.[9]

[9] Thus, "relational goal" should perhaps be renamed into "hornish goal", and "functional goal" into "footed goal". However, this would entail new words in the later definitions for "relational"/"functional" derivation, answer, etc.

3 Relational-Functional Interpretations and Models

First, we will consider general interpretations of full first-order relational-functional languages. Then, these will be restricted to Herbrand-like interpretations of RELFUN's clause programs. Since the basic RELFUN formalized here does not contain a negation construct, we will neglect RELFUN's three-valued open-world semantics and its differentiation of the truth values false and unknown [3].

Definition 11 *A* pre-interpretation *J of a first-order relational-functional language L consists of:*

1. *A non-empty set D, called the* domain *of the pre-interpretation.*
2. *For each constant in L, the assignment of an element in D.*
3. *For each n-ary constructor in L, the assignment of a mapping from D^n to D.*

Definition 12 *An* interpretation *I of a first-order relational-functional language L consists of a pre-interpretation J with domain D of L together with:*

1. *For each n-ary relation symbol in L, the assignment of a mapping from D^n into $\{true, false\}$ (or, equivalently, a relation on D^n).*
2. *For each n-ary function symbol in L, the assignment of a mapping from D^n to 2^D, the powerset of D.*

We say I is based on *J.*

Definition 13 *Let J be a pre-interpretation of a first-order relational-functional language L. A* variable assignment (wrt J) *is an assignment to each variable in L of an element in the domain of J.*

Definition 14 *Let J be a pre-interpretation with domain D of a first-order relational-functional language L and let V be a variable assignment. The* term assignment (wrt J and V) *of the terms in L is defined as follows:*

1. *Each variable is given its assignment according to V.*
2. *Each constant is given its assignment according to J.*
3. *If k' is the assignment of the n-ary constructor k according to J and $t'_1, \ldots, t'_n$ are the term assignments of $t_1, \ldots, t_n$, then $k'(t'_1, \ldots, t'_n) \in D$ is the term assignment of $k[t_1, \ldots, t_n]$.*

Definition 15 *Let I be an interpretation with domain D of a first-order relational-functional language L and let V be a variable assignment. The* expression assignment (wrt I and V) *of the expressions in L is defined as follows:*

1. *If t' is the term assignment of the term t wrt I and V, then $\{t'\}$ is the expression assignment of t.*

2. *If f' is the mapping assigned to the n-ary function symbol f by I and $E'_1, \ldots, E'_n$ are the expression assignments of $E_1, \ldots, E_n$, then the union of all $f'(t'_1, \ldots, t'_n) \in 2^D$ for each $t'_1 \in E'_1$, ..., $t'_n \in E'_n$ is the expression assignment of $f(E_1, \ldots, E_n)$.*

Definition 16 *Let I be an interpretation with domain D of a first-order relational-functional language L and let V be a variable assignment. Then a formula in L can be given a* truth value*, true or false, (wrt I and V) as follows (we let (a possibly embellished version of) t denote a term, of e, denote a flat application, of E, denote an expression, and of W, denote a formula):*

1. *If the formula has the form $r(E_1, \ldots, E_n)$, then the truth value of the formula is true if there exist $t'_1 \in E'_1$, ..., $t'_n \in E'_n$ such that $r'(t'_1, \ldots, t'_n)$ has truth value true, where r' is the mapping assigned to r by I and $E'_1, \ldots, E'_n$ are the expression assignments of $E_1, \ldots, E_n$ wrt I and V; otherwise, the formula's truth value is false.*

2. *If the formula has the form $f(t_1, \ldots, t_n)$ `:-&` E, then the truth value of the formula is true if the expression assignment of E wrt I and V is E' and $E' \subseteq f'(t'_1, \ldots, t'_n)$, where f' is the mapping assigned to f by I, and $t'_1, \ldots, t'_n$ are the term assignments of $t_1, \ldots, t_n$ wrt I and V; otherwise, the formula's truth value is false.*

3. *If the formula has the form t `is` E, then its truth value is true if the expression assignment of E wrt I and V is E' and $t' \in E'$, where t' is the term assignment of t wrt I and V; otherwise, its truth value is false.* [10]

4. *If the formula has the form e `:-` $false$ `&` E, then its truth value is true. If the formula has the form e `:-` $true$ `&` E, then the truth value is that of e `:-&` E.* [11]

5. *If the formula has the form $\neg W$, $W_1 \wedge W_2$, $W_1 \vee W_2$, W_1 `:-` W_2, or $W_1 \leftrightarrow W_2$, then the truth value is given by the usual truth tables.*

6. *If the formula has the form $\forall x W$, then the truth value of the formula is true if for all $d \in D$ the subformula W has truth value true wrt I and $V(x/d)$, where $V(x/d)$ is V except that x is assigned d; otherwise, the formula's truth value is false.*

[10] Thus the instance t `is` $f(t_1, \ldots, t_n)$ has the same truth value as the instance $f(t_1, \ldots, t_n)$ `:-&` t, defined through item *2.* The different syntaxes are maintained even in these special cases for marking off the body-goal use of the former from the clause-definition use of the latter. Also, in RELFUN's implemented operational semantics, successful setters return their evaluated rhs, rather than just `true`.

[11] For formalizing RELFUN's "valued conjunctions", definition 3 could introduce a third class of expressions, co-inductively with the formulas of definition 4, making the symbol "`&`" a binary infix instead of its actual use as part of a ternary mixfix: If W is a formula and E is an expression, then (W `&` E) is an expression. This enables simulating formulas of the form e `:-` W `&` E by nestings of the form e `:-&` (W `&` E). For this, the expression ($true$ `&` E) can be assigned the value of E. However, assigning $false$ to ($false$ `&` E), blurring the distinction between (2^D-valued) expressions and ($\{true, false\}$-valued) formulas, would, e.g., cause $fac(N)$ `:-&` ($zerop(N)$ `&` 1) to return $false$ for $fac(1)$ instead of signalling inapplicability. Therefore, in RELFUN ($false$ `&` E) is actually assigned the failure-signalling truth value *unknown*, which can be regarded as the empty expression value $\{\} \in 2^D$.

7. *If the formula has the form $\exists x W$, then its truth value is true if there exists $d \in D$ such that W has truth value true wrt I and $V(x/d)$; otherwise, its truth value is false.*

This functionally extended truth concept directly transfers to the classical definitions of, e.g., *model*, *validity*, and *logical consequence*, for which we refer to [7].

Example 4 *Consider the formula $\forall x(x \texttt{ is } f(g(x), g(x)))$ and the following interpretation I. Let $D = \{1, 2, \ldots\}$ be the natural numbers, let f be assigned the function that maps two naturals to the singleton set of their product, and let g be assigned the function that maps a natural to the set of its divisors. Then I is a model of the formula because all naturals have at least themselves and 1 as divisors.*

The definitions of groundness and Herbrand universes and bases adapt the corresponding classical notions; the definitions of Herbrand crosses and crossbases extend the notion of Herbrand bases in order to define models of, respectively, functional and relational-functional programs, as motivated in section 1.

Definition 17 *A* ground term, ground atom, *or* ground molecule *is, respectively, a term, atom, or molecule not containing variables.*

Definition 18 *The* Herbrand universe U_P *of a program P is the set of all ground terms that can be formed out of the constants and constructors appearing in P.*

Definition 19 *The* Herbrand base B_P *of a program P is the set of all ground atoms that can be formed by using the relation symbols from P with ground terms from the Herbrand universe U_P as arguments.*

Definition 20 *The* Herbrand cross C_P *of a program P is the set of all ground molecules that can be formed by using the function symbols from P with ground terms from the Herbrand universe U_P as arguments and using ground terms from U_P as foots.*

Definition 21 *The* Herbrand crossbase X_P *of a program P is the union $B_P \cup C_P$ of its Herbrand base B_P and its Herbrand cross C_P.*

Example 5 *The (deterministic, extra-variables, is-less) program P_1*

$f(X) \texttt{ :- } p(X), q(Y) \texttt{ \& } g(g(X,Y),Y).$
$g(a,a) \texttt{ :- \& } k[X].$
$g(k[X], l[X]) \texttt{ :- \& } g(X,X).$
$p(k[X]).$
$q(l[X]).$

uses the constructors k and l, and employs the operators f and g (as functions) as well as p and q (as relations).

The Herbrand universe U_{P_1} *of* P_1 *is*
$\{a, k[a], l[a], k[k[a]], k[l[a]], l[k[a]], l[l[a]], \ldots\}$.

The Herbrand base B_{P_1} *of* P_1 *is*
$\{p(a), q(a), p(k[a]), p(l[a]), q(k[a]), q(l[a]), \ldots\}$.

The Herbrand cross C_{P_1} *of* P_1 *is*
$\{f(a) \text{ :-@ } a, f(a) \text{ :-@ } k[a], f(a) \text{ :-@ } l[a], \ldots,$
$g(a,a) \text{ :-@ } a, g(a,a) \text{ :-@ } k[a], g(a,a) \text{ :-@ } l[a], \ldots,$
$\ldots\}$.

The Herbrand crossbase $X_{P_1} = B_{P_1} \cup C_{P_1}$ *of* P_1 *is*
$\{p(a), q(a), p(k[a]), p(l[a]), q(k[a]), q(l[a]), \ldots,$
$f(a) \text{ :-@ } a, f(a) \text{ :-@ } k[a], f(a) \text{ :-@ } l[a], \ldots,$
$g(a,a) \text{ :-@ } a, g(a,a) \text{ :-@ } k[a], g(a,a) \text{ :-@ } l[a], \ldots,$
$\ldots\}$.

Two generalized model concepts can now be defined, extending the usual Herbrand models for relational programs to models for functional and relational-functional programs.

Definition 22 *An* Herbrand (base), Herbrand cross, or Herbrand crossbase interpretation *is a subset of the Herbrand base, Herbrand cross, or Herbrand crossbase, respectively.*

Definition 23 *Let I be an Herbrand (base), Herbrand cross, or Herbrand crossbase interpretation and let P be a program. Then I is, respectively, an* Herbrand (base), Herbrand cross, or Herbrand crossbase model *for P if P is true wrt I.*

We concentrate the further development on relational-functional Herbrand crossbase models, which, however, constitute disjoint unions of Herbrand cross models and Herbrand (base) models.

The "model intersection" proposition 6.1 of [7] obviously also holds for the crossbase extension.

Proposition 1 (Model intersection property) *Let P be a relational-functional program and* $\{M_i\}_{i \in I}$ *be a non-empty set of Herbrand crossbase models for P. Then* $\bigcap_{i \in I} M_i$ *is an Herbrand crossbase model for P.*

Since every relational-functional program P has X_P as an Herbrand crossbase model, the set of all Herbrand crossbase models for P is non-empty, and proposition 1 permits the following definition.

Definition 24 *The* least Herbrand crossbase model M_P *for a relational-functional program P is the intersection of all Herbrand crossbase models for P.*

Example 6 *For u assuming all values from U_{P_1}, the following Herbrand crossbase interpretation I, contained in X_{P_1}, is an (the least) Herbrand crossbase model of P_1 (cf. example 5):*

$\{f(k[a])$ *:-&* $k[u]$, $g(a,a)$ *:-&* $k[u]$, $g(k[a],l[a])$ *:-&* $k[u]$,
$p(k[u])$, $q(l[u])\}$.

Thus, while P_1 deterministically returns the non-ground term $k[X]$ for certain arguments of the functions f and g (failing for other ones), the model of P_1 contains infinitely nondeterministic molecules that let f and g return the ground terms $k[a], k[k[a]], k[l[a]], \ldots$ for the same argument combinations.

Proposition 2 *Let P be a relational-functional program and I an Herbrand crossbase model of P (in particular, the least one). Then there exist a Horn program $\tilde{P}$ and an Herbrand model $\tilde{I}$ of $\tilde{P}$ (in particular, the least one) such that there is a bijection between I and $\tilde{I}$.*

Example 7 *The relational-functional program P_1 of example 5 can be transformed into the following Horn program $\tilde{P}_1$ by flattening the g nesting and introducing result parameters for f and g (note that the g-molecule becomes an atom):*

$\tilde{f}(X,R)$ *:-* $p(X), q(Y), \tilde{g}(X,Y,S), \tilde{g}(S,Y,R)$.
$\tilde{g}(a,a,k[X])$.
$\tilde{g}(k[X],l[X],R)$ *:-* $\tilde{g}(X,X,R)$.
$p(k[X])$.
$q(l[X])$.

An (the least) Herbrand model $\tilde{I}$ of $\tilde{P}_1$ is (where $u \in U_{\tilde{P}_1} = U_{P_1}$):

$\{\tilde{f}(k[a],k[u])$, $\tilde{g}(a,a,k[u])$, $\tilde{g}(k[a],l[a],k[u])$,
$p(k[u])$, $q(l[u])\}$.

The bijection between I and $\tilde{I}$ is obvious: untilded (functional) molecules correspond to tilded (relational) atoms; untilded atoms remain unchanged.

While the above bijection, call it b_{LAST}, introduces the new parameter in position $n+1$, there is another bijection, b_{FIRST}, introducing it in position 1, as actually done by RELFUN's `relationalize` *algorithm [3]. That is, an Herbrand model such as $\tilde{I}$ alone does not carry the entire information of the original Herbrand crossbase model such as I: the type of bijection must be specified along with the Herbrand model in order to preserve in the relations the computation direction ('mode') of the original functions. For instance, while $(b_{LAST})^{-1} \circ b_{LAST}(I) = I$, the composition $(b_{FIRST})^{-1} \circ b_{LAST}$ would transform I to*

the Herbrand crossbase model

$\{f(k[u])$:-& $k[a],\ \ g(a,k[u])$:-& $a,\ \ g(l[a],k[u])$:-& $k[a],$
$p(k[u]),\ \ q(l[u])\}$

which is not equivalent to I.

Let us now proceed to the generalized notions of relational-functional answers and their correctness.

Definition 25 *Let* P *be a relational-functional program and* G_r *and* G_f *be a relational and a functional goal, respectively. A* relational answer *for* $P \cup \{G_r\}$ *is a substitution for variables of* G_r. *A* functional answer *for* $P \cup \{G_f\}$ *is a term paired with a substitution for variables of* G_f.

It should be understood that the substitution does not necessarily contain a binding for every variable in G_r or G_f. Since RELFUN's operational semantics considers relations as true-valued functions, a relational answer operationally returns the term `true` along with yielding a substitution.

Definition 26 *Let* P *be a relational-functional program,* G_r *a relational goal* :- $B_1, \ldots, B_k$ *with* θ *an answer for* $P \cup \{G_r\}$, *and* G_f *a functional goal* :- $B_1, \ldots, B_k$ & F *with* (t, θ) *an answer for* $P \cup \{G_f\}$. *We say that* θ *is a* correct (relational) answer *for* $P \cup \{G_r\}$ *if* $\forall((B_1 \wedge \ldots \wedge B_k)\theta)$ *is a logical consequence of* P. *We say that* (t, θ) *is a* correct (functional) answer *for* $P \cup \{G_f\}$ *if* $\forall((B_1 \wedge \ldots \wedge B_k \wedge (t$ `is` $F))\theta)$ *is a logical consequence of* P.

The following lemma shows that functional answers, i.e. "value returning to the top-level", can be simulated by relational answers binding top-level return values to a special variable.

Lemma 1 *Let* P *be a relational-functional program,* G_f *a functional goal* :- $B_1, \ldots, B_k$ & F, *and* G_r *a relational goal* :- $B_1, \ldots, B_k, (x$ `is` $F)$ *with* x *a new variable. Then the following statements are equivalent:*

1. (t, θ) *is a correct functional answer for* $P \cup \{G_f\}$.
2. $\theta\{x/t\}$ *is a correct relational answer for* $P \cup \{G_r\}$.

Proof
(t, θ) *is a correct functional answer for* $P \cup \{G_f\}$
iff
$\forall((B_1 \wedge \ldots \wedge B_k \wedge (t$ `is` $F))\theta)$ *is a logical consequence of* P
iff
$\forall((B_1 \wedge \ldots \wedge B_k \wedge (x$ `is` $F))\theta\{x/t\})$ *is a logical consequence of* P
iff
$\theta\{x/t\}$ *is a correct relational answer for* $P \cup \{G_r\}$.

4 SLV-Resolution

We now extend SLD-resolution to first-order relational-functional clauses, where the SLD-case will be called *body resolution*. The extended resolution method, similar to innermost conditional narrowing [5], will be called *SLV-resolution* (SL-resolution for "Valued clauses" i.e., RELFUN's definite-clause extension). It provides the set of inference rules of a first-order relational-functional theory. The detailed example 8 at the end of this section will illustrate most SLV-resolution concepts.

Definition 27 *Let G_r be the relational goal* `:-` $B_1, \ldots, B_m, \ldots, B_k$*; further let C be the hornish clause d* `:-` $V_1, \ldots, V_v$ *or the footed clause e* `:-` $W_1, \ldots, W_w$ `&` E *or the trivial clause $\top$. Then G'_r is* (relationally) derived *from G_r and C using mgu θ if one of the following five inference rules applies (we let t's or u's denote terms):*

Body resolution

1. *B_m is an atom, called the* selected atom*, in G_r.*
2. *C is the hornish clause d* `:-` $V_1, \ldots, V_v$ *and θ is the mgu of B_m and d.*
3. *G'_r is the relational goal* `:-` $(B_1, \ldots, B_{m-1}, V_1, \ldots, V_v, B_{m+1}, \ldots, B_k)\theta$.

***is*-rhs resolution**

1. *B_m is a formula of the form t* `is` $g(u_1, \ldots, u_m)$*, called the* selected flat setter*, in G_r.*
2. *C is the footed clause e* `:-` $W_1, \ldots, W_w$ `&` E *and θ is the mgu of $g(u_1, \ldots, u_m)$ and e.*
3. *G'_r is the relational goal* `:-` $(B_1, \ldots, B_{m-1}, W_1, \ldots, W_w, t$ `is` $E, B_{m+1}, \ldots, B_k)\theta$.

Body flattening

1. *B_m in G_r is a formula of the form $r(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)$, called the* selected nested relationship*, and $h(E_{i,1}, \ldots, E_{i,n_i})$ is an embedded application, called the* selected (relationship-)embedded application.
2. *C is the trivial clause $\top$ and θ is the identity substitution (hence, trivially, an mgu).*
3. *x is a new variable.*
4. *G'_r is the relational goal* `:-` $B_1, \ldots, B_{m-1}$, x `is` $h(E_{i,1}, \ldots, E_{i,n_i}), r(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m), B_{m+1}, \ldots, B_k$.

***is*-rhs flattening**

1. *B_m in G_r is a formula of the form t* `is` $g(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)$*, called the* selected nested setter*, and $h(E_{i,1}, \ldots, E_{i,n_i})$ is an embedded application, called the* selected (`is`-)embedded application.

2. *C is the trivial clause $\top$ and θ is the identity substitution (hence, trivially, an mgu).*
3. *x is a new variable.*
4. *G'_r is the relational goal* $:- B_1, \ldots, B_{m-1}, x \texttt{ is } h(E_{i,1}, \ldots, E_{i,n_i}), t \texttt{ is } g(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m), B_{m+1}, \ldots, B_k$.

Term unification

1. *B_m is a formula of the form $t_1 \texttt{ is } t_2$, called the* selected term setter, *in G_r.*
2. *C is the trivial clause $\top$ and θ is the mgu of t_1 and t_2.*
3. *G'_r is the relational goal* $:- (B_1, \ldots, B_{m-1}, B_{m+1}, \ldots, B_k)\theta$.

Definition 28 *Let G_f be the functional goal* $:- B_1, \ldots, B_k \texttt{ \& } F$; *further let C be the hornish clause* $d :- V_1, \ldots, V_v$ *or the footed clause* $e :- W_1, \ldots, W_w \texttt{ \& } E$ *or the trivial clause $\top$. Then G'_f is* (functionally) derived *from G_f and C using mgu θ if one of the following three inference rules applies (we let u's denote terms):*

Relational subderivation (using one of the five rules of definition 27)

1. *G_r is* $:- B_1, \ldots, B_k$, *called the* selected relational subgoal *of G_f.*
2. *G'_r is relationally derived from G_r and C using mgu θ.*
3. *G'_f is the functional goal* $:- G'_r \texttt{ \& } F\theta$.

Foot resolution

1. *F is a formula of the form $g(u_1, \ldots, u_m)$, called the* selected flat application, *in G_f.*
2. *C is the footed clause $e :- W_1, \ldots, W_w \texttt{ \& } E$ and θ is the mgu of $g(u_1, \ldots, u_m)$ and e.*
3. *G'_f is the functional goal* $:- (B_1, \ldots, B_k, W_1, \ldots, W_w \texttt{ \& } E)\theta$.

Foot flattening

1. *F in G_f is a formula of the form $g(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)$, called the* selected nested application, *and $h(E_{i,1}, \ldots, E_{i,n_i})$ is an embedded application, called the* selected (application-)embedded application.
2. *C is the trivial clause $\top$ and θ is the identity substitution (hence, trivially, an mgu).*
3. *x is a new variable.*
4. *G'_f is the functional goal* $:- B_1, \ldots, B_k, x \texttt{ is } h(E_{i,1}, \ldots, E_{i,n_i}) \texttt{ \& } g(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m)$.

Although we first presented relational goals (in definition 27) and then extended them to functional goals (in definition 28), the inference rules would not have to distinguish body and foot premises for their "selection function" (or, item *1.* of each rule), and they do not in the actual implementation: (relational) body resolution and (functional) foot resolution, as well as body and foot flattening, could be treated together. Similarly, inference rules operating in the top-level of premises and in `is`-rhs's have a common realization: (relational) body resolution and (functional) `is`-rhs resolution, as well as body flattening and `is`-rhs flattening, could be identified. However, our more discriminative presentation will clarify the case analysis of the soundness proof.

Definition 29 *Let P be a relational-functional program and G be a (relational or functional) goal. A* (relational resp. functional) SLV-derivation *of $P \cup \{G\}$ consists of a finite or infinite sequence $G_0 = G, G_1, G_2, \ldots$ of (relational resp. functional) goals, a sequence $C_1, C_2, \ldots$ of variants of program clauses of $P \cup \{\top\}$, $\top$ the trivial clause, and a sequence $\theta_1, \theta_2, \ldots$ of mgu's such that each G_{i+1} is derived from G_i and C_{i+1} using θ_{i+1}.*

Definition 30 *A* (relational) SLV-refutation *of $P \cup \{G_r\}$, G_r a relational goal, is a finite SLV-derivation of $P \cup \{G_r\}$ that has the empty hornish clause as the last goal in the derivation. A* (functional) SLV-refutation *of $P \cup \{G_f\}$, G_f a functional goal, is a finite SLV-derivation of $P \cup \{G_f\}$ that has the terminal footed clause $\triangle(t)$ as the last goal in the derivation. If $G_n =$ or $G_n = \triangle(t)$, we say the refutation has* length n.

Definition 31 *An* unrestricted (relational or functional) SLV-refutation *is a (relational or functional) SLV-refutation, except that the substitutions θ_i are not required to be most general unifiers. They are only required to be unifiers.*

Definition 32 *Let P be a relational-functional program. The* relational success set *of P is the set of all ground atoms $a \in B_P$ such that $P \cup \{$ `:-` $a\}$ has a relational SLV-refutation. The* functional success set *of P is the set of all ground molecules $(e$ `:-&` $t) \in C_P$ such that $P \cup \{$ `:-&` $e\}$ has a functional SLV-refutation with last goal $\triangle(t)$. The* success set *of P is the union of the relational and functional success sets of P.*

Proposition 3 *Let P be a relational-functional program. The* functional success set *of P is the set of all ground molecules $(e$ `:-&` $t) \in C_P$ such that $P \cup \{$ `:-` $(t$ `is` $e)\}$ has a relational SLV-refutation.*

Proof
The ground flat setter $(t$ `is` $e) = (e$ `:-&` $t)^{\otimes}$ leads to a relational SLV-refutation iff e, also being the corresponding molecule's ground flat application, leads to a functional SLV-refutation with last goal $\triangle(t)$.

Definition 33 *Let P be a relational-functional program; further, let G_r be a relational goal. Suppose there is an SLV-refutation of $P \cup \{G_r\}$ and let $\theta_1, \ldots, \theta_n$ be its sequence of mgu's. A* computed (relational) answer *for $P \cup \{G_r\}$ is the substitution θ obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of G_r.*

Definition 34 *Let P be a relational-functional program; further, let G_f be a functional goal. Suppose there is an SLV-refutation of $P \cup \{G_f\}$ and let $\theta_1, \ldots, \theta_n$ be its sequence of mgu's and let $\triangle(t)$ be its last goal. A* computed (functional) answer *for $P \cup \{G_f\}$ is the pair $(t\theta_1 \ldots \theta_n, \theta)$, with the term t extracted from $\triangle(t)$ and the substitution θ obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of G_f.*

Lemma 2 *Let P be a relational-functional program, G_f a functional goal* `:-` $B_1, \ldots, B_k$ `&` F, *and G_r a relational goal* `:-` $B_1, \ldots, B_k, (x$ `is` $F)$ *with x a new variable. Then the following statements are equivalent:*

1. *(t, θ) is a computed functional answer for $P \cup \{G_f\}$.*
2. *$\theta\{x/t\}$ is a computed relational answer for $P \cup \{G_r\}$.*

Proof
(t, θ) is a computed functional answer for $P \cup \{G_f\}$
iff
there is an SLV-refutation of $P \cup \{G_f\}$ with a sequence of mgu's $\theta_1, \ldots, \theta_n$ and last goal $\triangle(u)$ such that t is $u\theta_1 \ldots \theta_n$ and θ restricts the composition $\theta_1 \ldots \theta_n$ to the variables of G_f
iff
there is an SLV-refutation of $P \cup \{G_r\}$ with a sequence of mgu's $\theta_1, \ldots, \theta_n, \{x/t\}$ such that $\theta\{x/t\}$ restricts the composition $\theta_1 \ldots \theta_n\{x/t\}$ to the variables of G_r
iff
$\theta\{x/t\}$ is a computed relational answer for $P \cup \{G_r\}$.

Example 8 *The (non-deterministic, no-extra-variables,* `is`*-using) program P_2*

$f(X)$ `:-` $p(g(a), g(X))$ `&` $h(g(X))$.
$g(a)$ `:-&` c.
$g(a)$ `:-&` $h(c)$.
$h(X)$ `:-&` b.
$p(X, c)$ `:-` X `is` $h(a), q(h(X))$.
$q(b)$.

uses no constructors, hence belongs to the DATALOG-extending DATAFUN subset of RELFUN; it has the finite Herbrand universe $\{a, b, c\}$, hence a finite Herbrand crossbase.

A functional SLV-refutation of $P_2 \cup \{$ `:-&` $f(Y)\}$ *is:*

$G_0 = G =$ `:-&` $f(Y)$
Foot resolution of $f(Y)$ with $C_1 = f(X1)$ `:-` $p(g(a), g(X1))$ `&` $h(g(X1))$, $\theta_1 = \{Y/X1\}$*:*
$G_1 =$ `:-` $p(g(a), g(X1))$ `&` $h(g(X1))$
Body flattening of $p(g(a), \ldots)$ with $C_2 = \top$, $\theta_2 = \{\}$:
$G_2 =$ `:-` $Z1$ `is` $g(a), p(Z1, g(X1))$ `&` $h(g(X1))$
`is`*-rhs resolution of $Z1$* `is` $g(a)$ *with $C_3 =$* $g(a)$ `:-&` $h(c)$, $\theta_3 = \{\}$*:*

$G_3 =$ *:-* $Z1$ *is* $h(c), p(Z1, g(X1))$ *&* $h(g(X1))$
is-rhs resolution of $Z1$ *is* $h(c)$ *with* $C_4 = h(X2)$ *:-&* b, $\theta_4 = \{X2/c\}$*:*
$G_4 =$ *:-* $Z1$ *is* $b, p(Z1, g(X1))$ *&* $h(g(X1))$
Term unification of $Z1$ *is* b *with* $C_5 = \top$, $\theta_5 = \{Z1/b\}$*:*
$G_5 =$ *:-* $p(b, g(X1))$ *&* $h(g(X1))$
Body flattening of $p(\ldots, g(X1))$ *with* $C_6 = \top$, $\theta_6 = \{\}$*:*
$G_6 =$ *:-* $Z2$ *is* $g(X1), p(b, Z2)$ *&* $h(g(X1))$
is-rhs resolution of $Z2$ *is* $g(X1)$ *with* $C_7 = g(a)$ *:-&* c, $\theta_7 = \{X1/a\}$*:*
$G_7 =$ *:-* $Z2$ *is* $c, p(b, Z2)$ *&* $h(g(X1))$
Term unification of $Z2$ *is* c *with* $C_8 = \top$, $\theta_8 = \{Z2/c\}$*:*
$G_8 =$ *:-* $p(b, c)$ *&* $h(g(X1))$
Body resolution of $p(b, c)$ *with* $C_9 = p(X3, c)$ *:-* $X3$ *is* $h(a), q(h(X3))$, $\theta_9 = \{X3/b\}$*:*
$G_9 =$ *:-* b *is* $h(a), q(h(b))$ *&* $h(g(X1))$
is-rhs resolution of b *is* $h(a)$ *with* $C_{10} = h(X4)$ *:-&* b, $\theta_{10} = \{X4/a\}$*:*
$G_{10} =$ *:-* b *is* $b, q(h(b))$ *&* $h(g(X1))$
Term unification of b *is* b *with* $C_{11} = \top$, $\theta_{11} = \{\}$*:*
$G_{11} =$ *:-* $q(h(b))$ *&* $h(g(X1))$
Body flattening of $q(h(b))$ *with* $C_{12} = \top$, $\theta_{12} = \{\}$*:*
$G_{12} =$ *:-* $Z3$ *is* $h(b), q(Z3)$ *&* $h(g(X1))$
is-rhs resolution of $Z3$ *is* $h(b)$ *with* $C_{13} = h(X5)$ *:-&* b, $\theta_{13} = \{X5/b\}$*:*
$G_{13} =$ *:-* $Z3$ *is* $b, q(Z3)$ *&* $h(g(X1))$
Term unification of $Z3$ *is* b *with* $C_{14} = \top$, $\theta_{14} = \{Z3/b\}$*:*
$G_{14} =$ *:-* $q(b)$ *&* $h(g(X1))$
Body resolution of $q(b)$ *with* $C_{15} = q(b)$, $\theta_{15} = \{\}$*:*
$G_{15} =$ *:-&* $h(g(a))$[12]
Foot flattening of $h(g(a))$ *with* $C_{16} = \top$, $\theta_{16} = \{\}$*:*
$G_{16} =$ *:-* $Z4$ *is* $g(a)$ *&* $h(Z4)$
is-rhs resolution of $Z4$ *is* $g(a)$ *with* $C_{17} = g(a)$ *:-&* c, $\theta_{17} = \{\}$*:*
$G_{17} =$ *:-* $Z4$ *is* c *&* $h(Z4)$
Term unification of $Z4$ *is* c *with* $C_{18} = \top$, $\theta_{18} = \{Z4/c\}$*:*
$G_{18} =$ *:-&* $h(c)$
Foot resolution of $h(c)$ *with* $C_{19} = h(X6)$ *:-&* b, $\theta_{19} = \{X6/b\}$*:*
$G_{19} =$ *:-&* b

This length-19 refutation happens to use RELFUN's implemented PROLOG-like 'leftmost' computation rule (however, RELFUN implements flattening in a condensed '`and`*-parallel' fashion). Operationally speaking, "*$f(Y)$ *returns* b *and binds* Y *to* a*": The refutation has last goal* $G_{19} = \triangle(b)$*, and* $\theta_1 \ldots \theta_{19}$ *restricted to* Y *is* $\{Y/a\}$*; hence the computed functional answer is* $(b, \{Y/a\})$*.*

The equivalent computed relational answer for $P_2 \cup \{$ *:-* Z *is* $f(Y)\}$ *is* $\{Y/a, Z/b\}$*. Here, the refutation uses* `is`*-rhs resolutions and performs an* `is`*-rhs flattening instead of the corresponding rules operating on the foot, and it needs a final term unification. Functional computation is somewhat hidden in the auxiliary setter's rhs. However, the kernel*

[12] The binding $\theta_7 = \{X1/a\}$ from the relational subderivation $G_2, \ldots, G_{15}$ is applied here.

subderivations of the functional and relational refutations are essentially the same.

The success set of P_2 is (functional and relational partitions displayed in separate lines):[13]

$\{f(a)$:-& $b,\ g(a)$:-& $b,\ g(a)$:-& $c,\ h(a)$:-& $b,\ h(b)$:-& $b,\ h(c)$:-& $b,$
$q(b),\ p(b,c)\}$

5 Soundness of SLV-Resolution

While the following result addresses relational goals, only the first of the five SLV-resolution rules to be considered corresponds to the classical case of logic programming as proved by K. L. Clark.

Theorem 1 (Soundness of relational SLV-resolution) *Let P be a relational-functional program and G_r a relational goal. Then every computed answer for $P \cup \{G_r\}$ is a correct answer for $P \cup \{G_r\}$.*

Proof
Let G_r be the relational goal :- $B_1, \ldots, B_k$ *and $\theta_1, \ldots, \theta_n$ be the sequence of mgu's used in an SLV-refutation of $P \cup \{G_r\}$. We have to show that $\forall((B_1 \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P. The result is proved by induction on the length of the refutation.*

Suppose first that $n = 1$. This means that G_r is a goal of the form :- B_1, *to which either of two of the five SLV-resolution rules applies:*

Body resolution *B_1 is an atom, the program has a unit clause of the form d* :- , *and $B_1\theta_1 = d\theta_1$. Since $B_1\theta_1$* :- *is an instance of a unit clause of P, it follows that $\forall(B_1\theta_1)$ is a logical consequence of P.*

is-rhs resolution *Cannot derive in one step.*

Body flattening *Cannot derive in one step.*

is-rhs flattening *Cannot derive in one step.*

Term unification *B_1 is a formula of the form t_1* is *t_2 and θ_1 is the mgu of t_1 and t_2. Since $t_1\theta_1 = t_2\theta_1$, it follows that $\forall(B_1\theta_1)$ is valid, hence, trivially, is a logical consequence of P.*

Next suppose that the result holds for computed answers that come from SLV-refutations of length $n-1$. Suppose $\theta_1, \ldots, \theta_n$ is the sequence of mgu's used in a refutation of $P \cup \{G_r\}$ of length n. One of the five SLV-resolution rules applies:

[13] In higher-order RELFUN, this can be obtained from the computed answers of an operator-variable, varying-arity goal [3] such as :-& $Op(|Args)$.

Body resolution *Let B_m be the selected atom of G_r and the hornish clause d :- $V_1, \ldots, V_v$ ($v \geq 0$) be the first input clause. By the induction hypothesis, $\forall((B_1 \wedge \ldots \wedge B_{m-1} \wedge V_1 \wedge \ldots \wedge V_v \wedge B_{m+1} \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P. Thus, if $v > 0$, $\forall((V_1 \wedge \ldots \wedge V_v)\theta_1 \ldots \theta_n)$ is a logical consequence of P. In this case, as well as for $v = 0$, $\forall(B_m\theta_1 \ldots \theta_n)$, which is the same as $\forall(d\theta_1 \ldots \theta_n)$, is a logical consequence of P. Hence $\forall((B_1 \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P.*

is-rhs resolution *Let B_m be the selected flat setter t `is` $g(u_1, \ldots, u_m)$ of G_r and the footed clause e :- $W_1, \ldots, W_w$ & E ($w \geq 0$) be the first input clause. By the induction hypothesis, $\forall((B_1 \wedge \ldots \wedge B_{m-1} \wedge W_1 \wedge \ldots \wedge W_w \wedge t$ `is` $E \wedge B_{m+1} \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P. Thus, for $w \geq 0$, $\forall((W_1 \wedge \ldots \wedge W_w \wedge t$ `is` $E)\theta_1 \ldots \theta_n)$ is a logical consequence of P. Consequently, $\forall(B_m\theta_1 \ldots \theta_n)$, which is the same as $\forall((t$ `is` $e)\theta_1 \ldots \theta_n)$, is a logical consequence of P. Hence $\forall((B_1 \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P.*

Body flattening *Let B_m be the selected nested relationship $r(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)$ with the selected embedded application $h(E_{i,1}, \ldots, E_{i,n_i})$ of G_r. By the induction hypothesis, $\forall((B_1 \wedge \ldots \wedge B_{m-1} \wedge (x$ `is` $h(E_{i,1}, \ldots, E_{i,n_i})) \wedge r(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m) \wedge B_{m+1} \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$, x the new variable chosen by the SLV-refutation, is a logical consequence of P. Thus, $\forall((x$ `is` $h(E_{i,1}, \ldots, E_{i,n_i}))\theta_1 \ldots \theta_n)$ and $\forall(r(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m)\theta_1 \ldots \theta_n)$ are logical consequences of P. Consequently, $\forall(B_m\theta_1 \ldots \theta_n)$ is a logical consequence of P. Hence $\forall((B_1 \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P.*

is-rhs flattening *Let B_m be the selected nested setter t `is` $g(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)$ with the selected embedded application $h(E_{i,1}, \ldots, E_{i,n_i})$ of G_r. By the induction hypothesis, $\forall((B_1 \wedge \ldots \wedge B_{m-1} \wedge (x$ `is` $h(E_{i,1}, \ldots, E_{i,n_i})) \wedge (t$ `is` $g(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m)) \wedge B_{m+1} \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$, x the new variable chosen by the SLV-refutation, is a logical consequence of P. Thus, $\forall((x$ `is` $h(E_{i,1}, \ldots, E_{i,n_i}))\theta_1 \ldots \theta_n)$ and $\forall((t$ `is` $g(E_1, \ldots, E_{i-1}, x, E_{i+1}, \ldots, E_m))\theta_1 \ldots \theta_n)$ are logical consequences of P. Consequently, $\forall(B_m\theta_1 \ldots \theta_n)$ is a logical consequence of P. Hence $\forall((B_1 \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P.*

Term unification *Let B_m be the selected term setter t_1 `is` t_2 of G_r. By the induction hypothesis, $\forall((B_1 \wedge \ldots \wedge B_{m-1} \wedge B_{m+1} \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P. Since $t_1\theta_1 \ldots \theta_n = t_2\theta_1 \ldots \theta_n$, it follows that $\forall(B_m\theta_1 \ldots \theta_n)$ is valid, hence, trivially, is a logical consequence of P. Hence $\forall((B_1 \wedge \ldots \wedge B_k)\theta_1 \ldots \theta_n)$ is a logical consequence of P.*

The result for relational goals naturally carries over to functional goals.

Corollary 1 (Soundness of functional SLV-resolution) *Let P be a relational-functional program and G_f a functional goal. Then every computed answer for $P \cup \{G_f\}$ is a correct answer for $P \cup \{G_f\}$.*

Proof
By lemmas 2 and 1 there is an equivalent relational goal with computed and correct answers for which the soundness result of theorem 1 holds.

Corollary 2 *The success set of a relational-functional program is contained in its least Herbrand crossbase model.*

Proof
Let the program be P and suppose $F \in X_P$ is in the success set of P. By proposition 3, the success set of P is the set of all $F \in X_P$ such that $P \cup \{ \texttt{:-}\ F^{\otimes}\}$ has a relational refutation. By theorem 1, $F^{\otimes}$, hence F, is a logical consequence of P. Thus, F is true wrt all Herbrand crossbase models of P, hence is in P's least Herbrand crossbase model.

6 Least Herbrand Crossbase Models as Fixpoints

We now define T_P-like immediate-consequence operators on Herbrand crossbase interpretations. For this we employ *unnesting* of clause premises, a fixpoint-semantics, ground-formula analogue to flattening in SLV-resolution. Instead of introducing new variables, unnesting chooses any ground terms from the Herbrand universe, as "returned values", to link the subformulas generated from the original formula.

Definition 35 *A set of* unnested setters $unnestis_P(t\ \texttt{is}\ E)$ *of a ground setter* $t\ \texttt{is}\ E$ *for a program P is defined recursively as the non-deterministic mapping*

$unnestis_P(t\ \texttt{is}\ g(u_1, \ldots, u_m)) =$
$\quad \{t\ \texttt{is}\ g(u_1, \ldots, u_m)\}\ if\ \{u_1, \ldots, u_m\} \subseteq U_P$
$unnestis_P(t\ \texttt{is}\ g(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)) =$
$\quad unnestis_P(u\ \texttt{is}\ h(E_{i,1}, \ldots, E_{i,n_i})) \cup unnestis_P(t\ \texttt{is}\ g(E_1, \ldots, E_{i-1}, u, E_{i+1}, \ldots, E_m))$
$\quad for\ some\ u \in U_P$

Definition 36 *A set of* unnested formulas $unnest_P(V)$ *of a ground relationship or setter V for a program P is defined as the non-deterministic mapping*

$unnest_P(r(u_1, \ldots, u_m)) =$
$\quad \{r(u_1, \ldots, u_m)\}\ if\ \{u_1, \ldots, u_m\} \subseteq U_P$
$unnest_P(r(E_1, \ldots, E_{i-1}, h(E_{i,1}, \ldots, E_{i,n_i}), E_{i+1}, \ldots, E_m)) =$
$\quad unnestis_P(u\ \texttt{is}\ h(E_{i,1}, \ldots, E_{i,n_i})) \cup unnest_P(r(E_1, \ldots, E_{i-1}, u, E_{i+1}, \ldots, E_m))$
$\quad for\ some\ u \in U_P$
$unnest_P(t\ \texttt{is}\ t) =$
$\quad \{\}\ if\ t \in U_P$
$unnest_P(t\ \texttt{is}\ g(E_1, \ldots, E_m)) =$
$\quad unnestis_P(t\ \texttt{is}\ g(E_1, \ldots, E_m))$

A first auxiliary immediate-consequence operator, TB_P, generates atoms from atoms and molecules.

Definition 37 *Let P be a relational-functional program. The mapping $TB_P : 2^{X_P} \to 2^{B_P}$ is defined as follows. Let $I \in 2^{X_P}$ be an Herbrand crossbase interpretation. Then:*

$$TB_P(I) = \{w \in B_P \mid \begin{array}{l} w \texttt{ :- } V_1, \ldots, V_n \textit{ is a ground instance of a clause in } P, \\ unnest_P(V_k)^{\otimes} \subseteq I \textit{ for } 1 \leq k \leq n\} \end{array}$$

If each V_k has the Horn-premise form $r(u_1, \ldots, u_n)$ of an atom, $unnest_P(V_k)^{\otimes}$ just denotes the unit set $\{V_k\}$, hence TB_P becomes the T_P operator of M. H. van Emden and R. Kowalski.

Proposition 4 *Let P be a relational-functional program containing Horn clauses only and $I \in 2^{B_P}$ be an Herbrand interpretation. Then the mapping TB_P restricted to $2^{B_P} \subseteq 2^{X_P}$ specializes to the mapping $T_P : 2^{B_P} \to 2^{B_P}$ defined as:*

$$T_P(I) = \{w \in B_P \mid \begin{array}{l} w \texttt{ :- } V_1, \ldots, V_n \textit{ is a ground instance of a clause in } P, \\ V_k \in I \textit{ for } 1 \leq k \leq n\} \end{array}$$

Note how the intuitive understanding of T_P is extended by TB_P: as $T_P(I)$ 'guesses' a ground clause of P and then checks whether its premise atoms are members of I, $TB_P(I)$ 'guesses' a ground clause of P, then 'guesses' an unnesting (zero/one atoms and one/zero or more setters) from each of its premises, and then checks whether the "$^{\otimes}$"-corresponding atoms and molecules constitute subsets of I.

A second auxiliary immediate-consequence operator, TC_P, generates molecules from atoms and molecules.

Definition 38 *Let P be a relational-functional program. The mapping $TC_P : 2^{X_P} \to 2^{C_P}$ is defined as follows. Let $I \in 2^{X_P}$ be an Herbrand crossbase interpretation. Then:*

$$TC_P(I) = \{e \texttt{ :-\&} t \in C_P \mid \begin{array}{l} e \texttt{ :- } V_1, \ldots, V_n \texttt{ \& } E \textit{ is a ground instance of a clause in } P, \\ unnest_P(V_k)^{\otimes} \subseteq I \textit{ for } 1 \leq k \leq n, \\ unnest_P(t \texttt{ is } E)^{\otimes} \subseteq I\} \end{array}$$

Example 9 *The program P_2 (cf. example 8) with $U_{P_2} = \{a, b, c\}$ contains the footed clause $f(X)$:- $p(g(a), g(X))$ & $h(g(X))$. Suppose a TC_{P_2} application selects the ground instance $f(a)$:- $p(g(a), g(a))$ & $h(g(a))$, i.e. $V_1 = p(g(a), g(a))$ and $E = h(g(a))$. Then $unnest_{P_2}(V_1)$ can select $\{p(b, c),$ b is $g(a),$ c is $g(a)\}$, so that $unnest_{P_2}(V_1)^{\otimes} = \{p(b, c),$ $g(a)$:-& $b,$ $g(a)$:-& $c\}$. Further suppose TC_{P_2}'s set formation selects $t = b$ and $unnest_{P_2}(t$ is $E)$ selects $\{b$ is $h(c),$ c is $g(a)\}$, so that $unnest_{P_2}(t$ is $E)^{\otimes} = \{h(c)$:-& $b,$ $g(a)$:-& $c\}$. Now, if some interpretation I has $\{p(b, c),$ $g(a)$:-& $b,$ $g(a)$:-& $c,$ $h(c)$:-& $b\}$ as a subset, $TC_{P_2}(I)$ will contain the element $f(a)$:-& b.*

Since the sets produced by unnesting are always finite, the atoms and setters resulting from $unnest_P(V_k)$ and $unnest_P(t \texttt{ is } E)$ can be regarded as premises of a 'virtual' ground clause e :- $unnest_P(V_1)^\epsilon, \ldots, unnest_P(V_n)^\epsilon, unnest_P(t \texttt{ is } E)^\epsilon$ & t. ("$\{...\}^\epsilon$" denotes the sequence of *elements* of "{...}".) The corresponding non-ground clause can be obtained by transforming the original program P via static flattening and denotative normalization [2]. Therefore, each application of TC_P can be regarded as a condensed form of the application of a less powerful operator indexed by the more lengthy transformed program (T_P's extension would be confined to clauses with atomic and flat-setter bodies and term foots).

Example 10 *A virtual ground clause of* $f(a)$:- $p(g(a), g(a))$ & $h(g(a))$ *from example 9 is* $f(a)$:- b is $g(a), c$ is $g(a), p(b, c), c$ is $g(a), b$ is $h(c)$ & b. *Its non-ground abstraction* $f(X)$:- $Y1$ is $g(a), Y2$ is $g(X), p(Y1, Y2), Y3$ is $g(X), Y4$ is $h(Y3)$ & $Y4$ *is the flattened, denotative normalization of* $f(X)$:- $p(g(a), g(X))$ & $h(g(X))$, *the original non-ground clause.*

The main immediate-consequence operator, TX_P, just unites the two auxiliary ones.

Definition 39 *Let P be a relational-functional program. The mapping $TX_P : 2^{X_P} \to 2^{X_P}$ is defined as follows. Let $I \in 2^{X_P}$ be an Herbrand crossbase interpretation. Then:*

$$TX_P(I) = TB_P(I) \cup TC_P(I)$$

Example 11 *Let P_1 be the relational-functional program of example 5 and I the interpretation* {g(k[a],l[a]) :-& k[a], p(k[a]), q(l[a])} $\in 2^{X_{P_1}}$. *Since* $unnest_{P_1}(k[a]$ is $g(g(k[a], l[a]), l[a]))^\otimes$ *can select* {$g(k[a], l[a])$:-& $k[a]$}, *we obtain* $TX_{P_1}(I)$ = {f(k[a]) :-& k[a], g(a,a) :-& k[u], p(k[u]), q(l[u])} *for* $u \in U_{P_1}$.

Clearly, TX_P is monotonic on the complete lattice 2^{X_P} under the partial order "$\subseteq$". Like T_P in [7], it can be shown to be continuous.

Proposition 5 *Let P be a relational-functional program. Then the mapping TX_P is continuous.*

Proof
Let S be a directed subset of 2^{X_P}, V_k be a ground relationship or setter, for $1 \leq k \leq n$, and t is E be a ground setter. Each $unnest_P(V_k)^\otimes$ being a finite set, we can first note that $\bigcup_{k=1}^n unnest_P(V_k)^\otimes \subseteq lub(S)$ iff $\bigcup_{k=1}^n unnest_P(V_k)^\otimes \subseteq I$ for some $I \in S$; furthermore, $unnest_P(t \texttt{ is } E)^\otimes$ being a finite set, $\bigcup_{k=1}^n unnest_P(V_k)^\otimes \cup unnest_P(t \texttt{ is } E)^\otimes \subseteq lub(S)$ iff $\bigcup_{k=1}^n unnest_P(V_k)^\otimes \cup unnest_P(t \texttt{ is } E)^\otimes \subseteq I$ for some $I \in S$. In order to show that TX_P is continuous we have to show $TX_P(lub(S)) = lub(TX_P(S))$ for each directed subset S. Since TX_P denotes the disjoint union of TB_P's and TC_P's values we show the equality of both subsets individually:

$w \in TB_P(lub(S))$
iff
w *:-* $V_1, \ldots, V_n$ *is a ground instance of a clause in* P *and* $\bigcup_{k=1}^{n} unnest_P(V_k)^{\otimes} \subseteq lub(S)$
iff
w *:-* $V_1, \ldots, V_n$ *is a ground instance of a clause in* P *and* $\bigcup_{k=1}^{n} unnest_P(V_k)^{\otimes} \subseteq I$ *for some* $I \in S$
iff
$w \in TB_P(I)$ *for some* $I \in S$
iff
$w \in lub(TB_P(S))$

e *:-&* $t \in TC_P(lub(S))$
iff
e *:-* $V_1, \ldots, V_n$ *&* E *is a ground instance of a clause in* P *and* $\bigcup_{k=1}^{n} unnest_P(V_k)^{\otimes} \cup unnest_P(t$ *is* $E)^{\otimes} \subseteq lub(S)$
iff
e *:-* $V_1, \ldots, V_n$ *&* E *is a ground instance of a clause in* P *and* $\bigcup_{k=1}^{n} unnest_P(V_k)^{\otimes} \cup unnest_P(t$ *is* $E)^{\otimes} \subseteq I$ *for some* $I \in S$
iff
e *:-&* $t \in TC_P(I)$ *for some* $I \in S$
iff
e *:-&* $t \in lub(TC_P(S))$

Herbrand crossbase models can be characterized in terms of TX_P.

Proposition 6 *Let* P *be a relational-functional program and* I *be an Herbrand crossbase interpretation of* P. *Then* I *is a crossbase model for* P *iff* $TX_P(I) \subseteq I$.

Proof
I *is a crossbase model for* P
iff
for each ground instance w *:-* $V_1, \ldots, V_n$ *or* e *:-* $V_1, \ldots, V_n$ *&* E *of each clause in* P *we have, respectively,* $\bigcup_{k=1}^{n} unnest_P(V_k)^{\otimes} \subseteq I$ *implies* $w \in I$ *or* $\bigcup_{k=1}^{n} unnest_P(V_k)^{\otimes} \cup unnest_P(t$ *is* $E)^{\otimes} \subseteq I$ *implies* e *:-&* $t \in I$
iff
$TX_P(I) \subseteq I$

Using these propositions and general fixpoint results, we can extend the fixpoint characterization of the least Herbrand model of logic programs by M. H. van Emden and R. Kowalski to a characterization of the least Herbrand crossbase model of relational-functional programs (for the "$\uparrow$"-notation see [7]).

Theorem 2 (Fixpoint characterization of the least Herbrand crossbase model) *Let* P *be a relational-functional program. Then* $M_P = lfp(TX_P) = TX_P \uparrow \omega$.

Proof

$$\begin{aligned} M_P &= glb\{I \mid I \text{ is an Herbrand crossbase model for } P\} \\ &= glb\{I \mid TX_P(I) \subseteq I\}, \text{ by proposition 6} \\ &= lfp(TX_P), \text{ by proposition 5.1 in [7]} \\ &= TX_P \uparrow \omega, \text{ by proposition 5.4 in [7] and proposition 5} \end{aligned}$$

Example 12 *The 8-element least Herbrand crossbase model of the program P_2 of example 8 (in section 4) can be computed bottom-up by the following TX_{P_2} iterations (details of the last step were shown in example 9):*

$TX_{P_2} \uparrow 0 =$
$\{\}$

$TX_{P_2} \uparrow 1 = TX_{P_2} \uparrow 0 \ \cup$
$\{g(a) \ \text{:-\&} \ c, \ \ h(a) \ \text{:-\&} \ b, \ \ h(b) \ \text{:-\&} \ b, \ \ h(c) \ \text{:-\&} \ b,$
$q(b)\}$

$TX_{P_2} \uparrow 2 = TX_{P_2} \uparrow 1 \ \cup$
$\{g(a) \ \text{:-\&} \ b,$
$p(b, c)\}$

$M_{P_2} = lfp(TX_{P_2}) = TX_{P_2} \uparrow \omega = TX_{P_2} \uparrow 3 = TX_{P_2} \uparrow 2 \ \cup$
$\{f(a) \ \text{:-\&} \ b\}$

This is equal to the success set of P_2 given in example 8.

7 Completeness of SLV-Resolution

Like for soundness, we will again use proposition 3 as well as lemmas 1 and 2; hence the following mgu and lifting lemmas will only be needed for relational goals. The symbol "$\stackrel{G}{=}$" will denote equality between substitutions after restriction of the rhs substitution to the variables of the goal G.

Lemma 3 (Mgu lemma) *Let P be a relational-functional program and G_r a relational goal. Suppose that $P \cup \{G_r\}$ has an unrestricted SLV-refutation. Then $P \cup \{G_r\}$ has an SLV-refutation of the same length such that, if $\theta_1, \ldots, \theta_n$ are the unifiers from the unrestricted SLV-refutation and $\theta'_1, \ldots, \theta'_n$ are the mgu's from the SLV-refutation, then there exists a substitution γ such that $\theta_1 \ldots \theta_n \stackrel{G_r}{=} \theta'_1 \ldots \theta'_n \gamma$.*

Proof
The induction proof is as for lemma 8.1 in [7] except that unifiers and mgu's need not derive from (body) resolution but can derive from the other rules of SLV-resolution (flattening in unrestricted SLV-refutations, like in SLV-refutations, produces identity substitutions).

Lemma 4 (Lifting lemma) *Let P be a relational-functional program, G_r a relational goal, and θ a substitution. Suppose there exists an SLV-refutation of $P \cup \{G_r\theta\}$. Then there exists an SLV-refutation of $P \cup \{G_r\}$ of the same length such that, if $\theta_1, \ldots, \theta_n$ are the mgu's from the SLV-refutation of $P \cup \{G_r\theta\}$ and $\theta'_1, \ldots, \theta'_n$ are the mgu's from the SLV-refutation of $P \cup \{G_r\}$, then there exists a substitution γ such that $\theta\theta_1 \ldots \theta_n \stackrel{G_r}{=} \theta'_1 \ldots \theta'_n\gamma$.*

Proof
The proof is as for lemma 8.2 in [7] with the qualification already noted for lemma 3, which is crucially applied here.

The converse of corrollary 2 extends the logic-programming completeness result of K. R. Apt and M. H. van Emden to relational-functional programming.

Theorem 3 *The success set of a relational-functional program is equal to its least Herbrand crossbase model.*

Proof
Let the program be P. By corrollary 2 it suffices to show that the least Herbrand crossbase model of P is contained in the success set of P. Let F denote the ground atom d or molecule f `:-&` t. By proposition 3 we need only consider the relational goals denoted by $F^{\otimes}$. Suppose F is in the least Herbrand crossbase model of P. By theorem 2, $F \in TX_P \uparrow n$ for some $n \in \omega$. We prove by induction on n that $F \in TX_P \uparrow n$ implies that $P \cup \{$ `:-` $F^{\otimes}\}$ has a refutation (i.e., $d \in TX_P \uparrow n$ implies that $P \cup \{$ `:-` $d\}$ has a refutation and f `:-&` $t \in TX_P \uparrow n$ implies that $P \cup \{$ `:-` t `is` $f\}$ has a refutation). Hence F will be in the success set.
Suppose first that $n = 1$. Then $F \in TX_P \uparrow 1$ means that F is a ground instance of an atom or molecule from P. Clearly, $P \cup \{$ `:-` $d\}$ and $P \cup \{$ `:-` t `is` $f\}$ have a refutation (a body resolution and an `is`*-rhs resolution followed by a term unification, respectively).*
Now suppose that the result holds for $n-1$. We distinguish the two cases for F.
First, let $d \in TX_P \uparrow n$. By the definition of TX_P there exists a ground instance of a clause w `:-` $V_1, \ldots, V_m$ and an unnesting of its premises such that $d = w\theta$ and $\bigcup_{k=1}^{m} unnest_P(V_k\theta)^{\otimes} \subseteq TX_P \uparrow (n-1)$ for some unifier θ. By the induction hypothesis, for each formula A in the selected $unnest_P(V_k\theta)$, for $1 \leq k \leq m$, $P \cup \{$ `:-` $A\}$ has a refutation. Hence, $P \cup \{$ `:-` $V_k\theta\}$ has a refutation, mimicking unnesting by flattening. Because each $V_k\theta$ is ground and flattening only introduces new variables, these refutations can be combined into a refutation of $P \cup \{$ `:-` $(V_1, \ldots, V_m)\theta\}$. Thus $P \cup \{$ `:-` $d\}$ has an unrestricted refutation and we can apply the mgu lemma to obtain a refutation of $P \cup \{$ `:-` $d\}$.
Second, let f `:-&` $t \in TX_P \uparrow n$. By the definition of TX_P there exists a ground instance of a clause e `:-` $V_1, \ldots, V_m$ `&` E and an unnesting of its premises such that $f = e\theta$ and $\bigcup_{k=1}^{m} unnest_P(V_k\theta)^{\otimes} \cup unnest_P(t$ `is` $E\theta)^{\otimes} \subseteq TX_P \uparrow (n-1)$ for some unifier θ. By the induction hypothesis, for each formula A in the selected $unnest_P(V_k\theta)$, for $1 \leq k \leq m$, and $unnest_P(t$ `is` $E\theta)$, $P \cup \{$ `:-` $A\}$ has a refutation. Hence, $P \cup \{$ `:-` $V_k\theta\}$ and $P \cup \{$ `:-` t `is` $E\theta\}$ have a refutation, mimicking unnesting by flattening. Because each $V_k\theta$ and t `is` $E\theta$ are ground and flattening only introduces new variables, these refutations can be combined into a refutation of $P \cup \{$ `:-` $(V_1, \ldots, V_m, t$ `is` $E)\theta\}$. Thus $P \cup \{$ `:-` t `is` $f\}$

has an unrestricted refutation and we can apply the mgu lemma to obtain a refutation of $P \cup \{$ `:-` t `is` $f\}$.

For proving that every correct (relational or functional) answer is an instance of a computed (relational or functional) answer we first transfer lemma 8.5 from [7].

Lemma 5 *Let P be a relational-functional program and F a relationship or setter. Suppose that $\forall(F)$ is a logical consequence of P. Then there exists an SLV-refutation of $P \cup \{$ `:-` $F\}$ with the identity substitution as the computed answer.*

Proof
Suppose F has variables $x_1, \ldots, x_n$, anywhere in the relationship or on both sides of the setter. Let $a_1, \ldots, a_n$ be distinct constants not appearing in P or F and let θ be the substitution $\{x_1/a_1, \ldots, x_n/a_n\}$. Then it is clear that $F\theta$ is a logical consequence of P. Also, $F\theta$ being ground, each formula A in some $unnest_P(F\theta)$ is a logical consequence of P. Since each A is ground, theorem 3 shows that $P \cup \{$ `:-` $A\}$ has a refutation. Thus, $P \cup \{$ `:-` $F\theta\}$ has a refutation, mimicking unnesting by flattening. Since flattening only introduces new variables and the a_i do not appear in P or F, by replacing a_i by x_i, for $1 \leq i \leq n$, in this refutation, we obtain a refutation of $P \cup \{$ `:-` $F\}$ with the identity substitution as the computed answer.

Now, K. L. Clark's completeness result can be extended from logic to relational-functional programming. For relational goals we can adapt the formulation for definite goals in [7].

Theorem 4 (Completeness of relational SLV-resolution) *Let P be a relational-functional program and G_r a relational goal. For every correct answer θ for $P \cup \{G_r\}$ there exists a computed answer σ for $P \cup \{G_r\}$ and a substitution γ such that $\theta \stackrel{G_r}{=} \sigma\gamma$.*

Proof
Let the relational goal G_r be `:-` $B_1, \ldots, B_k$. *Since θ is correct, $\forall((B_1 \wedge \ldots \wedge B_k)\theta)$ is a logical consequence of P. By lemma 5 there exists a refutation of $P \cup \{$ `:-` $B_i\theta\}$ such that the computed answer is the identity, for $1 \leq i \leq k$. We can combine these refutations into a refutation of $P \cup \{$ `:-` $G_r\theta\}$ such that the computed answer is the identity.*
Suppose the sequence of mgu's of the refutation of $P \cup \{$ `:-` $G_r\theta\}$ is $\theta_1, \ldots, \theta_n$. Then $G_r\theta\theta_1 \ldots \theta_n = G_r\theta$. By the lifting lemma there exists a refutation of $P \cup \{$ `:-` $G_r\}$ with mgu's $\theta'_1, \ldots, \theta'_n$ such that $\theta\theta_1 \ldots \theta_n \stackrel{G_r}{=} \theta'_1 \ldots \theta'_n\gamma'$, for some substitution γ'. Let σ be $\theta'_1 \ldots \theta'_n$ restricted to the variables in G_r. Then $\theta \stackrel{G_r}{=} \sigma\gamma$, where γ is an appropriate restriction of γ'.

Again, the result for relational goals naturally carries over to functional goals.

Corollary 3 (Completeness of functional SLV-resolution) *Let P be a relational-functional program and G_f a functional goal. For every correct answer (t, θ) for $P \cup \{G_f\}$ there exists a computed answer (s, σ) for $P \cup \{G_f\}$ and a substitution γ such that $\theta \stackrel{G_f}{=} \sigma\gamma$ and $t = s\gamma$.*

Proof
By lemmas 1 and 2 there is an equivalent relational goal with correct and computed answers for which the completeness result of theorem 4 holds.

References

[1] M. Bellia and G. Levi. The relation between logic and functional languages: A survey. *Journal of Logic Programming*, 3:217–236, 1986.

[2] Harold Boley. A relational/functional language and its compilation into the WAM. Technical Report SEKI SR-90-05, University of Kaiserslautern, Department of Computer Science, April 1990.

[3] Harold Boley. Extended Logic-plus-Functional Programming. In *Workshop on Extensions of Logic Programming, ELP '91, Stockholm1991*, LNAI. Springer, 1992.

[4] D. DeGroot and G. Lindstrom, editors. *Logic Programming: Functions, Relations, and Equations.* Prentice-Hall, 1986.

[5] Laurent Fribourg. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *1985 Symposium on Logic Programming*, pages 172–184. IEEE Computer Society Press, 1985.

[6] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel-LEAF: A logic plus functional language. *Journal of Computer and System Sciences*, 42:139–185, 1991.

[7] John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[8] J.J. Moreno-Navarro and M. Rodriguez-Artalejo. Logic programming with functions and predicates: The language BABEL. *Journal of Logic Programming*, 12:191–223, 1992.

[9] M. J. O'Donnell. *Equational Logic as a Programming Language.* MIT Press, Cambridge, Mass., 1985.

[10] David H. D. Warren. Higher-order extensions to PROLOG: Are they needed? *Machine Intelligence*, 10:441–454, 1982.

Contents