# A Relational Characterization of Exhaustive Search and Backtracking

**Marcelo F. Frias***
Department of Computer Science
University of Buenos Aires
Argentina
mfrias@dc.uba.ar

**Gabriel A. Baum**
L.I.F.I.A.
School of Informatics
National University of La Plata
Argentina
gbaum@sol.info.unlp.edu.ar

**Nazareno M. Aguirre**
Department of Computer Science
National University of Rio Cuarto
Argentina
naguirre@dc.exa.unrc.edu.ar

**Abstract**

In previous papers we have presented a relational calculus for program construction based on fork algebras. This calculus allows the specification of algorithm design strategies as for instance case-analysis, trivialization and divide-and-conquer. In this paper we present an algebraic formalization of exhaustive search and backtracking, which is illustrated with an example.

## 1 Introduction

Fork algebras have been used as the foundation of a calculus for program construction since the late 80's. The calculus has gone through several stages: from using binary relations as a method to capture the input-output semantics of programs [9], to the calculus we shortly describe here and that possess the following features:

*Partially supported by Research Grant 16-2000 from LIFIA, School of Informatics, University of La Plata, Argentina.

- the calculus includes a highly declarative and expressive specification language.
- the completeness of the calculus implies excellent heuristic power.
- program design strategies are specified *within* the calculus.
- the mathematical background required to master the process of program development is minimal.

The calculus includes three languages:

- a language for problem specification,
- a language for program derivation, and
- a language for specification of program design strategies.

When using the three languages it is possible to derive algorithms from declarative specifications following previously determined design strategies. The paper is structured as follows. In Section 2 we introduce the specification language. In Section 3 we introduce the program derivation language. In Section 4 we present the language for formalization of algorithm design strategies, as well as some examples. In Section 5 we present the main results of the paper, namely, the formalizations of exhaustive search and backtracking, and we derive a backtracking solution for a class of problems. Finally, in Section 6 we present the conclusions and proposals for future work.

## 2 The Specification Language

**Starting with an Example** Let us consider the following first-order specification of the problem (binary predicate) $P \subseteq state \times listr(state)$ of finding a sequence of moves ending on a goal state in a board game.

$$P(x,l) \iff Hd(l) = x \ \wedge \ Goal\,(Lst(l)) \ \wedge \ \forall a,b\,(\exists l_1,l_2\,(l = l_1 ab l_2) \ \Rightarrow \ valid(a,b))\,, \qquad (1)$$

where $Hd$ and $Lst$ retrieve the first and the last element of a sequence, the predicate $valid$ determines if a change in the state of affairs in the game is valid or is a cheat, and the unary predicate $Goal$ characterizes the goal (or winning) states.

We can transform (1) into the following specification that uses binary relations and individuals:

$$\begin{aligned} x\,P\,l \iff l\,Hd\,x \ \wedge \ \exists c\,(l\,Lst\,c \ \wedge \ c\,Goal\,c) \ \wedge \\ \forall a,b\,(\exists l_1,l_2\,((l_1,l_2)\ Conc\,l \ \wedge \ l_1\,Lst\,a \ \wedge \ l_2\,Hd\,b) \ \Rightarrow \ (a,b)\,valid\,(a,b))\,, \qquad (2)\end{aligned}$$

where $Goal$ is a a filter (i.e., a binary relation included in the identity binary relation) that characterizes the goal states, $valid$ is a filter on those pairs $(a,b)$ for which there is a valid transition (move) from state $a$ to state $b$, and $Conc$ is concatenation of lists.

Notice that in the previous formula we have both relations and individual variables. Actually, the derivation calculus to be presented in Section 3 is dummy variables free, so there should be some technique allowing to get rid of dummy variables. Notice that the specification language to be presented in the following paragraphs includes an equational fragment which is free of variables ranging over individuals.

**Individual Terms** Let *IndVar* denote the set of *individual variables*. The set of *individual terms* (*IndTerm*) is the closure of *IndVar* under a pairing operation $(\ ,\ )$. Examples of individual terms are $(l_1, l_2)$ (that can be found in (2)), or $((x, y), z)$.

**Relation Designations** Let $\mathit{ExtRel} = \{ \mathit{Hd}, \mathit{Tl}, \mathit{Lst}, \mathit{Cons}, \mathit{Conc}, \ldots \}$ denote the set of *extralogical relation symbols*. The set of relation designations (*RelDes*) is the closure of $\mathit{ExtRel} \cup \{ 0, 1, 1' \}$ under the unary operators $^{-}$ and $\breve{}$, and the binary operators $+, \cdot, ;$ and $\nabla$.

- 0 is called the *empty* relation.
- 1 is called the *unit* relation.
- 1' is called the *identity* relation.
- $\overline{R}$ is read the *complement* of $R$.
- $\breve{R}$ is read the *converse* of $R$.
- $R{+}S$ is read the *join* or *union* of $R$ and $S$.
- $R{\cdot}S$ is read the *meet* or *intersection* of $R$ and $S$.
- $R;S$ is read the *relative product* or *composition* of $R$ and $S$.
- $R\nabla S$ is read the *fork* of $R$ and $S$.

**Atomic Formulas** Atomic formulas are expressions of the form $t_1\, R t_2$ (where $t_1$, $t_2$ are arbitrary individual terms and $R$ is an arbitrary relation designation) or $R = S$ (with $R$ and $S$ arbitrary relation designations).

**Compound Formulas** From the atomic formulas we obtain compound formulas as usual in classical first-order logic, by closing the atomic formulas under the logical constants $\neg$, $\forall$, $\exists$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$.

**Axioms and Inference Rules** We will choose a standard set of logical axioms and inference rules for classical first-order logic. As the axioms that explain the meaning of the symbols 0, 1, 1', $^{-}$, $\breve{}$, $+$, $\cdot$, ;, $\nabla$ and $(\ ,\ )$, we single out the following formulas, in which $x$, $y$, $z$, $u$, $v$ are arbitrary individual variables and $R$, $S$, $T$ are arbitrary relation designations.

$$
\begin{array}{ll}
\forall x \forall y\, (x\, 1\, y) & \forall x \forall y\, (\neg x\, 0\, y) \\
\forall x\, (x\, 1'\, x) & \forall x \forall y \forall z\, ((x\, R y \ \wedge \ y\, 1'\, z) \Rightarrow x\, R z) \\
\forall x \forall y \left( x\, \overline{R} y \ \Leftrightarrow \ \neg x\, R y \right) & \forall x \forall y \left( x\, \breve{R} y \ \Leftrightarrow \ y\, R x \right) \\
\forall x \forall y\, (x\, R{+}S y \ \Leftrightarrow \ x\, R y \vee x\, S y) & \forall x \forall y\, (x\, R{\cdot}S y \ \Leftrightarrow \ x\, R y \wedge x\, S y) \\
\forall x \forall y\, (x\, R;S y \ \Leftrightarrow \ \exists z\, (x\, R z \wedge z\, S y)) & R = S \ \Leftrightarrow \ \forall x \forall y\, (x\, R y \Leftrightarrow x\, S y)
\end{array}
$$

$$
\begin{array}{l}
\forall x \forall y\, (x\, R \nabla S y \ \Longleftrightarrow \ \exists u \exists v\, (y = (u, v) \ \wedge \ x\, R u \ \wedge \ x\, S v))\,, \\
\forall x \forall y \forall u \forall v\, ((x, y) = (u, v) \ \Rightarrow \ x = u \ \wedge \ y = v)\,.
\end{array}
$$

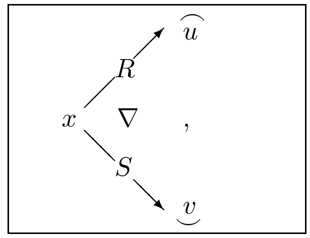


Figure 1: The operator *fork*.

**Semantics of the Specification Language** Notice that the language is fully determined by the set of extralogical relation symbols *ExtRel*. Similarly, structures for these languages are fully determined by

1. a nonempty set $U$ which is the domain of individuals,
2. the meaning of the pairing function $(\ ,\ ) : U \times U \to U$, and
3. the meaning of the extralogical relation symbols as binary relations on $U$.

If $m$ denotes the meaning function for relation designations, then:

$$\begin{array}{lcll}
m(0) & = & \emptyset, & \\
m(1) & = & U \times U, & \\
m(1') & = & \{\, \langle u, u \rangle : u \in U \,\}\,, & \\
m\left(\overline{R}\right) & = & (U \times U) \setminus m(R), & \text{(the complement of the meaning of } R) \\
m(\breve{R}) & = & \{\, \langle v, u \rangle : \langle u, v \rangle \in m(R) \,\}\,, & \\
m(R+S) & = & m(R) \cup m(S), & \\
m(R \cdot S) & = & m(R) \cap m(S), & \\
m(R;S) & = & m(R) \circ m(S), & \text{(the composition of the binary relations } m(R) \text{ and } m(S)) \\
m(R \nabla S) & = & \{\, \langle a, (b,c) \rangle : \langle a, b \rangle \in m(R) \wedge \langle a, c \rangle \in m(S) \,\}\,. &
\end{array}$$

A graphical interpretation of the operator fork is given in Fig. 1.

Valuations of the individual variables extend to individual terms as follows. If $\nu$ is a valuation of the individual variables, then $\nu((t_1, t_2)) = (\nu(t_1), \nu(t_2))$.

Given a structure $\mathcal{A}$ and a valuation of the individual terms $\nu$, for atomic formulas we have:

$$\mathcal{A} \models t_1 R t_2 \iff \langle \nu(t_1), \nu(t_2) \rangle \in m(R), \qquad \mathcal{A} \models R = S \iff m(R) = m(S)\ .$$

From the semantics of atomic formulas, the semantics of compound formulas is defined as is usually done in first-order logic.

**Fork Algebras of Binary Relations** If we are only concerned about the semantics of formulas of the form $S = T$, then it is enough to consider algebraic structures of the form

$$\langle\, R, \cup, \cap, ^{-}, \emptyset, E, ;, Id, ^{\smile}, \nabla \,\rangle$$

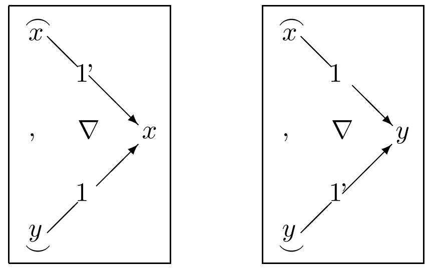

Figure 2: The projections $\pi$ and $\rho$.

such that

1. $R$ is set of binary relations on a set $U$ closed under a pairing operation ( , ).
2. $\bigcup R \subseteq E$,
3. If we denote by $Id$ the identity relation on the set $U$, then $\emptyset$, $E$ and $Id$ belong to $R$,
4. $R$ is closed under set union ($\cup$), intersection ($\cap$), complement relative to $E$ ($^{-}$), composition of binary relations ($\circ$), converse ($\breve{}$), and fork, the last being defined by the condition

$$S \nabla T = \{ \langle x, (y, z) \rangle : x\, S\, y \ \wedge \ x\, T\, z \} \ .$$

These algebras are called fork algebras of binary relations.

**Some Definable Constants and Operations** The constants $\pi = (1\text{'} \nabla 1)^{\smile}$ and $\rho = (1 \nabla 1\text{'})^{\smile}$ behave as projections, projecting components from pairs $(a, b)$. See Fig. 2 for a graphical interpretation.

We also define two more operations on binary relations called *cross* and *relational implication*. The operation *cross*, given a pair of binary relations performs a kind of parallel product. A graphic representation of cross is given in Fig. 3. Its set theoretical definition is given by

$$R \otimes S = \{\langle (x, y)\,, (u, v) \rangle : x\, R\, u \ \wedge \ y\, S\, v\}.$$

It is not difficult to check that cross is definable from the other relational operators with the use of fork. It is a trivial exercise to show that $R \otimes S = (\pi ; R) \nabla (\rho ; S)$.

The abstract definition of the relational implication of relations $R$ and $S$ is given by the equality

$$R \rightarrow S = \overline{R ; \breve{\overline{S}}},$$

while its set theoretical definition (see Fig. 3 for a graphical interpretation) is given by

$$R \rightarrow S = \{(x, y) : \forall z\, (x\, R\, z \Rightarrow y\, S\, z)\}.$$

The pair $(x, y)$ belongs to the relation $R \rightarrow S$ if the image set of $x$ via $R$ (represented by the small circle inside $C$) is contained in the image set of $y$ via $S$, the last represented by the medium sized circle inside $C$.

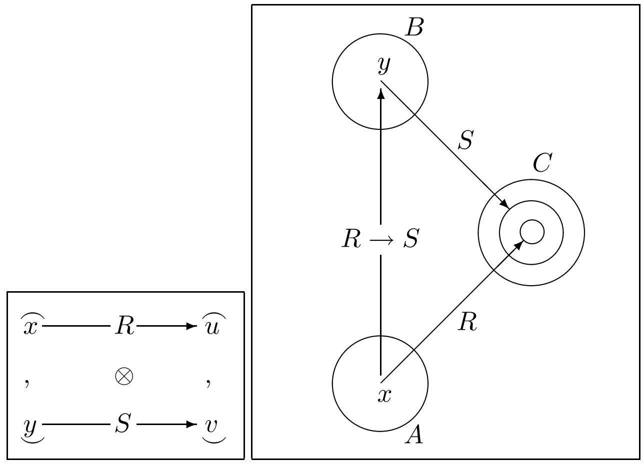


Figure 3: The operations *cross* and *relational implication*.

**Crashing Dummies** Going back to formula (2), we had

$$x\,P\,l \iff l\,Hd\,x \;\wedge\; \exists c\,(l\,Lst\,c \;\wedge\; c\,Goal\,c) \;\wedge$$
$$\forall a,b\,(\exists l_1,l_2\,((l_1,l_2)\;Concl \;\wedge\; l_1\,Lst\,a \;\wedge\; l_2\,Hd\,b) \;\Rightarrow\; (a,b)\;valid\,(a,b)) \;\;.$$

Let us show how to get rid of dummy variables in the previous formula.

1.

$$\begin{array}{ll} & l\,Hd\,x \\ \equiv & \{ \text{ by def. } {}^{\smile} \text{ } \} \\ & x\,Hd^{\smile}\;l\;. \end{array}$$

2.

$$\begin{array}{ll} & \exists c(l\,Lst\,c \;\wedge\; c\,Goal\,c) \\ \equiv & \{ \text{ by def. } ; \text{ } \} \\ & \exists c\,(l\,Lst;Goal\,c) \\ \equiv & \{ \text{ by def. } 1 \text{ } \} \\ & \exists c\,(l\,Lst;Goal\,c \;\wedge\; c1x) \\ \equiv & \{ \text{ by def. } ; \text{ } \} \\ & l\,Lst;Goal;1x \\ \equiv & \{ \text{ by def. } {}^{\smile} \text{ } \} \\ & x\,(Lst;Goal;1)^{\smile}l \\ \equiv & \{ \text{ by prop. } {}^{\smile} \text{ } \} \\ & x\,1;Goal;Lst^{\smile}\;l\;. \end{array}$$

3.

$$
\begin{array}{ll}
 & \exists l_1, l_2\,((l_1,l_2)\ Conc\,l\ \wedge\ l_1\,Lst\,a\ \wedge\ l_2\,Hd\,b) \\
\equiv & \{\ \text{by def. } \breve{}\ \text{and def. } \otimes\ \} \\
 & \exists l_1, l_2\,(l\,Conc^{\smile}\ (l_1,l_2)\ \wedge\ (l_1,l_2)\ Lst\otimes Hd\ (a,b)) \\
\equiv & \{\ \text{by def. } ;\ \} \\
 & l\,Conc^{\smile}\ ; (Lst\otimes Hd)\ (a,b)\ .
\end{array}
$$

4.

$$
\begin{array}{ll}
 & \forall a,b\,(l\,Conc^{\smile}\ ; (Lst\otimes Hd)\ (a,b)\ \Rightarrow\ (a,b)\ valid\ (a,b)) \\
\equiv & \{\ \text{by def. } 1\ \} \\
 & \forall a,b\,(l\,Conc^{\smile}\ ; (Lst\otimes Hd)\ (a,b)\ \Rightarrow\ x1\,(a,b)\ \wedge\ (a,b)\ valid\ (a,b)) \\
\equiv & \{\ \text{by def. } \rightarrow\ \} \\
 & l\,Conc^{\smile}\ ; (Lst\otimes Hd)\ \rightarrow\ 1;valid x \\
\equiv & \{\ \text{by def.}^{\smile}\ \} \\
 & x\,(Conc^{\smile}\ ; (Lst\otimes Hd)\ \rightarrow\ 1;valid)^{\smile} l\ .
\end{array}
$$

Joining items 1 – 4 above, we obtain

$$P = Hd^{\smile}\ \cdot 1;Goal;Lst^{\smile}\ \cdot (Conc^{\smile}\ ; (Lst\otimes Hd)\ \rightarrow\ 1;valid)^{\smile}\ . \tag{3}$$

Formula (3) is an equational specification of $P$ which is dummy variable free. This procedure can be carried on for arbitrary specifications containing dummy variables, as shows the following theorem:

Theorem 2.1 ***Veloso - Haeberer (1991)****[11] There exists a recursive mapping $T$ from formulas in the specification language to equations (no dummy variables) such that*

$$\models_{Spec} \alpha \quad \Longleftrightarrow \quad \vdash_{Spec} T(\alpha)\ .$$

# 3 The Equational Derivation Language

In this section is presented an equational formalism (based on the equational theory of fork algebras [4, 8]) for program derivation. In order to present the formalism (that we will call here *Deriv*), we will provide the formulas, axioms and inference rules.

**Formulas** Formulas will be equations between relational designations, thus, the set of formulas is the restriction of the formulas in *Spec* to the equational fragment. For instance, formula (3) is a formula in the derivation formalism.

**Axioms** The set of formulas chosen as axioms is given by:

A set of equation stating that $+$, $\cdot$, $^{-}$, 0 and 1 are the join, meet, complement, zero and unit in a Boolean algebra, respectively. To these formulas we add the following:

$$x;(y;z) = (x;y);z, \tag{Ax. 1}$$

$$(x+y)\,;z = x;z \;+\; y;z, \tag{Ax. 2}$$

$$(x+y)^{\smile} = \breve{x}+\breve{y}, \tag{Ax. 3}$$

$$\breve{\breve{x}} = x, \tag{Ax. 4}$$

$$x;1\text{'} = 1\text{'}\,;x = x, \tag{Ax. 5}$$

$$(x;y)^{\smile} = \breve{y};\breve{x}, \tag{Ax. 6}$$

$$x;y \,\cdot\, z = 0 \text{ iff } z;\breve{y} \,\cdot\, x = 0 \text{ iff } \breve{x};z \,\cdot\, y = 0. \tag{Ax. 7}$$

$$r\nabla s = (r;\breve{\pi})\cdot(s;\breve{\rho})\,, \tag{Ax. 8}$$

$$(r\nabla s)\,;(t\nabla q)^{\smile} = \big(r;\breve{t}\big)\cdot(s;\breve{q})\,, \tag{Ax. 9}$$

$$\pi\nabla\rho \;+\; 1\text{'} = 1\text{'}. \tag{Ax. 10}$$

Notice that Ax. 7 is not an equation, but it is equivalent to one.

**Inference Rules** Standard set of rules for equational logic (reflexivity, symmetry, transitivity, substitution and replacement).

**Abstract Fork Algebras** The class of abstract fork algebras is defined from the calculus as those algebraic structures $\langle\, R, +, \cdot, ^{-}, 0, 1, ;, 1\text{'}, ^{\smile}, \nabla \,\rangle$ which are models of the axioms.

## 3.1 A Bit on Notation and Arithmetical Properties

**Lists, their operations and properties** In the remaining sections we will use operations on lists. We will denote the domain of lists whose objects are from a domain $A$ by $listr(A)$. $[\,]$ is the empty list, and the list whose first element is $x$ and whose tail is $xs$ is noted by $[x : xs]$. The relation *Cons*, given a pair $(x, xs)$, generates the list $[x : xs]$. Given $[x : xs]$, the relation *Hd* returns $x$ and *Tl* returns $xs$. The relation *Lst* returns the last element of a nonempty list. Given a pair of lists $(l, l')$, the relation *Conc* returns their concatenation. Given a list $[x : x' : xs]$, with at least two elements, the relation *Snd* returns $x'$. The relation *pair* is defined as $Hd\nabla Snd$, and given a list $[x : x' : xs]$ with at least two elements, returns the pair $(x, x')$. Given $x \in A$, the relation *wrap* returns the list $[x]$. Conversely, given the list $[x]$, the relation *unwrap* returns the element $x$. The relation *App*, given a pair $(x, xs)$, appends the element $x$ to the end of list $xs$. We single out a few filters over relations. By $L^k$ $(k \in \mathbb{N})$ we denote the filter over lists of length $k$. Also, $L^{>k}$ is the filter over lists of length greater than $k$.

The following properties of the operations on lists will be used in derivations. Along a derivation the use of these properties will be marked by the keyword “list”.

$$
\begin{array}{lcll}
Dom\,(pair) & = & L^{>1}, & \\
L^{>1};Lst & = & Tl;Lst, & \\
unwrap & = & L^{1};Lst, & \\
Cons & = & (Hd \nabla Tl)^{\smile}, & \\
Hd^{\smile} \cdot 1;G;wrap & = & G;wrap & (G \text{ a filter}) \\
Snd & = & Tl;Hd &
\end{array}
$$

**Filters and their properties** Filters are relations contained in the identity relation. A special case are those filters that are described as the domain (or range) of relations. We will need the following properties of filters, referred to in the derivations by the keyword “filter promotion”.

$$
\begin{array}{lcll}
f;r \cdot s & = & f;(r \cdot s) & (f \text{ filter}) \\
f;r \nabla s & = & f;(r \nabla s) & (f \text{ filter}) \\
r \nabla f;s & = & f;(r \nabla s) & (f \text{ filter}) \\
\neg f & = & \overline{f;1} \cdot 1' & (f \text{ filter}) \\
\overline{f;1} & = & \neg f;1 & (f \text{ filter})
\end{array}
$$

We will also need the following properties about domains. Their use will be marked “*Dom*”.

$$
\begin{array}{lcl}
Dom\,(r) & = & r;1 \cdot 1' \\
Dom\,(r) & \leq & 1' \\
\neg Dom\,(r) & = & \overline{r;1} \cdot 1' \\
\neg Dom\,(r)\,;1 & = & \overline{r;1}
\end{array}
$$

**Functional relations and their properties** Functional relations are characterized by the property $\breve{f};f \leq 1'$. The following properties involve functional relations and the ways in which they distribute over the relational operators. Their use will be marked by the keyword “function promotion”.

$$
\begin{array}{lcll}
f;(r \cdot s) & = & f;r \cdot f;s, & (f \text{ functional}) \\
(r \cdot s)\,;\breve{f} & = & r;\breve{f} \cdot s;\breve{f}, & (f \text{ functional}) \\
f;\overline{r} & = & Dom\,(f)\,;\overline{f;r} & (f \text{ functional}) \\
\breve{f} \cdot 1;\left(\breve{f} \cdot s\right) & = & \breve{f} \cdot s & (f \text{ functional}) \\
f;r \cdot s & = & f;(r \cdot s)\,, & (f \text{ functional and } s \text{ left-ideal, i.e., } 1;s = s)
\end{array}
$$

**Properties of fork** We start by defining the relation $\mathit{2}$ as $1' \nabla 1'$. This relation, given the input $x$, produces the pair $(x, x)$ as output. The derivations require the following properties of the fork operator. We will refer to these properties by the name “fork”.

$$
\begin{array}{lcl}
2\,;\pi & = & 1' \\
2\,;\rho & = & 1' \\
(r\,\nabla\, s)\,;\pi & = & Dom\,(s)\,;r \leq r \\
(r\,\nabla\, s)\,;\rho & = & Dom\,(r)\,;s \leq s
\end{array}
$$

**Properties of the relational implication** The following properties hold about the relational implication, and will be labeled in proofs as "implication":

$$
\begin{array}{lcl}
r+s \;\rightarrow\; t & = & (r \rightarrow t) \cdot (s \rightarrow t) \\
f \rightarrow r & = & f\,;\breve{r} \,+\, \neg Dom\,(f)\,;1
\end{array}
$$

LEMMA 3.1 *Let:* $F := f1 \,+\, f2;F$, $d1 := Dom\,(f1)$, $d2 := Dom\,(f2)$, $r2 := Ran\,(f2)$, $d3 := Dom\,(f2;(r2-d2))$, *and* $g := d2;\,(F \rightarrow r)$. *Assume that (i)* $d1 \leq d2 \leq r2$, *and (ii)* $(r2-d2)\cdot d1 = 0$. *Then* $g = f1;\breve{r} \,\cdot\, f2;g \;+\; d3;f1;\breve{r} \;+\; (d2-d1)\,;f2;g \;+\; (d3\cdot\,(d2-d1))\,;1$ .

*Proof.*

$$
\begin{array}{cc}
 & g \\
= & \{\ \text{unfold } g \text{ and } F\ \} \\
 & d2;\,(f1 \,+\, f2;F \;\rightarrow\; r) \\
= & \{\ \text{by implication and filter promotion}\ \} \\
 & d2;\,(f1 \rightarrow r) \;\cdot\; d2;\,(f2;F \;\rightarrow\; r) \\
= & \{\ \text{by implication and (i)}\ \} \\
 & (f1;\breve{r} \,+\, (d2-d1);1) \;\cdot\; f2;\,(F \rightarrow r) \\
= & \{\ \text{by Boolean algebra}\ \} \\
 & (f1;\breve{r} \,+\, (d2-d1);1) \;\cdot\; f2;\,(d2+r2-d2)\,;\,(F \rightarrow r) \\
= & \{\ \text{by Boolean algebra}\ \} \\
 & (f1;\breve{r} \,+\, (d2-d1);1) \;\cdot\; (f2;d2;\,(F \rightarrow r) \;+\; f2;(r2-d2);\,(F \rightarrow r)) \\
= & \{\ \text{by Boolean algebra, (ii) and fold } g\ \} \\
 & (f1;\breve{r} \,+\, (d2-d1);1) \;\cdot\; (f2;g \;+\; f2;(r2-d2);1) \\
= & \{\ \text{by def. } d3\ \} \\
 & (f1;\breve{r} \,+\, (d2-d1);1) \,\cdot\, (f2;g \;+\; d3;1) \\
= & \{\ \text{by Boolean algebra and filter promotion}\ \} \\
 & f1;\breve{r} \,\cdot\, f2;g \;+\; d3;f1;\breve{r} \;+\; (d2-d1);f2;g \;+\; d3\cdot(d2-d1);1\ .
\end{array}
$$

■

## 3.2 Program Derivation, Representability, Completeness and Expressiveness

The process of program derivation is quite elementary. Starting from a declarative specification in the specification formalism, this specification is then transformed into an equational (but nonalgorithmic) specification using Thm. 2.1. From the equational specification, we can start applying the rules of the formalism *Deriv* until transforming the equational specification into a new equational specification that resembles an algorithm. This has been used for instance in [1, 7].

Let us consider now the following situation, in which we are given a specification $\alpha$. By Thm. 2.1 $\models_{Spec} \alpha \iff \vdash_{Spec} T(\alpha)$. Let us suppose that $T(\alpha)$ has the shape $L = R$, with $L$ and $R$ relation designations. Let us suppose that a relation designation $R'$ defines the same binary relation as $R$. Then $\vdash_{Spec} R = R'$.

This can be considered as a heuristic coming from the standard semantics of the calculus. If the calculus is capable of proving $\vdash_{Deriv} R = R'$, then we can say that the calculus profits from these heuristics, or that the calculus has a good heuristic power. Since the axioms defining the meaning of operators in the derivation formalism are sound with respect to the binary relations semantics, it is immediate that $\vdash_{Deriv} R = R' \Rightarrow \vdash_{Spec} R = R'$.

On the other hand, good heuristic power amounts to the completeness of the derivation formalism with respect to the specification formalism. The completeness is an immediate consequence of the following representation theorem for fork algebras.

THEOREM 3.2 ***Frias-Haeberer-Veloso (1995)****[6] Every abstract fork algebra is isomorphic to a fork algebra of binary relations.*

Regarding the expressiveness of the calculus, from the representation and completeness theorems follows the following theorem.

THEOREM 3.3 ***Frias - Baum (1995)****[2] Given a formula $\alpha$ in the specification language,*

$$\models_{Spec} \alpha \quad \iff \quad \vdash_{Deriv} T(\alpha) \ .$$

Notice that Thm. 3.3 is stronger than Thm. 2.1. Thm. 3.3 allows to express specifications as equations in the derivation formalism, while Thm. 2.1 allows to do so only in the specification formalism.

# 4 Adding Program Design Strategies

In [3, 5, 8] we extended the derivation formalism in order to include the definition of program design strategies as for instance case analysis, trivialization, divide and conquer, and others. The design strategies are expressed using the first-order language of fork algebras. A trivial example of a design strategy is case analysis ($C\_A$). A problem is said to be solved using this strategy if the domain of the problem can be partitioned, let us say, in $k$ parts $D_1, \ldots, D_k$, and we find $k$ algorithms $A_1, \ldots, A_k$ such that $A_i$ solves the given problem when its domain is restricted to the part $D_i$. This can be more simply and formally stated by the following formula over relations:

$$C\_A(R, R_1, \ldots, R_k) \iff \bigwedge_{1 \le i < j \le k} Dom\,(R_i) \cdot Dom\,(R_j) = 0 \ \wedge \ R = \sum_{i=1}^{k} R_i.$$

**Remark** Notice that from Thm. 3.2, the classes of fork algebras of binary relations and abstract fork algebras are elementarily equivalent, i.e., share the same first-order theory. Therefore, formulas from the first-order theory of fork algebras have a standard semantics in terms of binary relations.

The strategy of trivialization ($Triv$) is a particular instance of case analysis where one of the subproblems is assumed to be easy to solve. Easy subproblems are for example those whose solution does not depend on the original problem (non-recursive parts), or for which a solution is at hand. We then have

$$Triv(R, R_0, R_1, \ldots, R_k) \iff C\_A(R, R_0, R_1, \ldots, R_k) \wedge Easy(R_0).$$

The relations $R_0, R_1, \ldots, R_k$ are usually determined by properties of the problem domain. In general, domains allow for "natural" partitions (empty and nonempty lists, trees of height 1 or greater, etc.).

The strategy of recomposition (*Recomp*) is defined by the formula

$$Recomp(R, Split, Q_1, \ldots, Q_k, Join) \iff R = Split;(Q_1 \otimes \cdots \otimes Q_k);Join,$$

where the relations *Split* and *Join* stand for programs so that the first one effectively decomposes the data, and the latter combines the results of $Q_1, \ldots, Q_k$ in order to provide a solution for $R$.

By joining the strategies of recomposition and trivialization, we obtain the following formalization of divide-and-conquer:

$$D\&C(R, R_0, Split, Q_1, \ldots, Q_k, Join) \iff \\ \exists Q\, (Triv(R, R_0, Q) \wedge Recomp(Q, Split, Q_1, \ldots, Q_k, Join)),$$

where the variable $R$ may appear inside some of the terms $Q_1, \ldots, Q_k$, but does not affect the term $R_0$.

Generally, relations $Q_1, \ldots, Q_k$ will be either 1' or the relation $R$ itself. This is supported for example by Smith [10] in his schema of divide-and-conquer algorithms.

It is easy to see how the previously given strategies induce the structure of the programs. For example, it is clear from the definition of case analysis that whenever $C\_A(R, R_0, R_1)$ holds, we can infer that $R = R_0 + R_1$, thus giving a program (equation) of the desired shape solving the problem $R$. In the same way, when $D\&C(R, R_0, Split, Q_1, \ldots, Q_k, Join)$ holds, we have a program with shape $R = R_0 + Split;(Q_1 \otimes \cdots \otimes Q_k);Join$ solving $R$.

Notice that design strategies are in general described by formulas of the form

$$Strat(R, X_1, \ldots, X_n) \qquad \iff \qquad Strat\_Definition(R, X_1, \ldots, X_n),$$

where *Strat* is a predicate symbol and *Strat_Definition* is a formula (involving the relational variables $R$, $X_1, \ldots, X_n$) that defines the strategy *Strat*.

# 5 Formalizing Exhaustive Search and Backtracking

We formalize a nondeterministic version of the exhaustive search algorithm by the formula

$$Ex-Search(ES, m_1, \ldots, m_k, G, S) \iff S \leq 1' \wedge \bigwedge_{1 \leq i \leq k} m_i{}^{\smile};m_i \leq S \wedge G \leq S \wedge \\ ES = G;wrap + \neg G;\sum_{i=1}^{k} \begin{pmatrix} 1' \\ \nabla \\ m_i;ES \end{pmatrix};Cons. \quad (4)$$

Formula (4) is to be read as follows:

Given a set of *goal* states $G$ (a filter), and functional relations $m_1, \dots, m_k$ denoting valid transitions between states from the set of states $S$ ($S$ a filter), the problem $ES$ has an exhaustive search solution characterized by the functions $m_1, \dots, m_k$, $G$ and $S$ if the equality

$$ES = G;wrap \; + \; \neg G;\sum_{i=1}^{k}\left(\begin{array}{c} 1' \\ \nabla \\ m_i;ES \end{array}\right);Cons$$

is true.

Let us consider the following procedural schema of the backtracking algorithm that returns a path to a goal state in the state space tree (if there is any), and halts if there in no path in the tree reaching a goal state.

```
Function Backtracking(state s) : listr(state)
    l := Backtracking'([s]);
    if Goal(Lst(l)) then
       return(l)
    else
       Halt
    fi
end
```

where the function Backtracking' is given by:

```
Function Backtracking'(listr(state) l) : listr(state)
    l' := l;
    i := 1;
    while i ≤ #moves ∧ not Goal(Lst(l')) do
            if feasible(l, m_i) then
              l' := Backtracking' ([l : m_i(Lst(l))])
            fi
            i := i + 1;
    od
    return(l')
end
```

In order to obtain a relational version of the previous algorithm, we will model the finite loop in the while statement by means of a composition of relations. We will denote by $B'_k$ the Backtracking' relation for a $k$ moves problem, and by $B_k$ its associated Backtracking function.

Let $(F_i)_{1 \le i \le k}$ be the family of binary relations defined by

$$F_i := Dom\left(\pi;f_i\right);\pi;\left(\begin{array}{c} 1' \\ \nabla \\ \left(Lst;m_i \;\nabla\; 1'\right);App;B'_k \end{array}\right) \; + \; \neg Dom\left(\pi;f_i\right) \; .$$

for each $i$, $1 \le i \le k$, $F_i$ stands for the **if** statement inside the **while** loop.

Let the family of relations $(M_i)_{k\geq i\geq 1}$ be defined by the conditions:

$$M_k := \neg Dom\,(\rho;Lst;G)\,;F_k \;+\; Dom\,(\rho;Lst;G)\,,$$

For $1 \leq j < k$, we define

$$M_j := \neg Dom\,(\rho;Lst;G)\,;F_j\,;M_{j+1} \;+\; Dom\,(\rho;Lst;G)\ \ .$$

Notice that relation $M_1$ stands for the **while** loop.

Finally, the relation $B_k$ is defined by the two conditions

$$B'_k = 2\,;M_1\,;\rho,$$

$$B_k = wrap;B'_k\,;Dom\,(Lst;G)\ \ .$$

We then specify the backtracking strategy by the formula

$$Backtracking(P, m_1, f_1, \ldots, m_k, f_k, G, S) \iff \\ G \leq S \;\wedge\; \bigwedge_{1\leq i\leq k} m_i{}^{\smile}\;;m_i \leq S \;\wedge\; \bigwedge_{1\leq i\leq k} f_i \leq 1\text{'} \;\wedge\; P = B_k\ . \quad (5)$$

## 5.1 Example

In this subsection we will present an example illustrating how to use the previous specifications of the exhaustive search and the backtracking algorithms in order to solve problems. Earlier in this paper, from formula (2) from the specification formalism, we obtained the following equation in the derivation formalism:

$$P = Hd^{\smile}\;\cdot\;1;G;Lst^{\smile}\;\cdot\;(Conc^{\smile}\;;(Lst\otimes Hd)\;\rightarrow\;1;valid)^{\smile}\ .$$

In Appendix A, we present a derivation showing that

$$P = G;wrap \;+\; \neg G;\sum_{i=1}^{k}\left(\begin{array}{c} 1\text{'} \\ \nabla \\ m_i\,;P \end{array}\right);Cons, \quad (6)$$

where we assume that $valid = \sum_{i=1}^{k} v_i$ and each $v_i$ represents the transition of states produced by an atomic valid movement in the space state, and define $m_i := \breve{\pi}\,;v_i\,;\rho$. Notice that $m_i$ is the actual transition described by the filter $v_i$.

We have therefore proved the following theorem:

THEOREM 5.1

$$P = Hd^{\smile}\;\cdot\;1;G;Lst^{\smile}\;\cdot\;(Conc^{\smile}\;;(Lst\otimes Hd)\;\rightarrow\;1;valid)^{\smile} \\ \wedge\;valid = \sum_{i=1}^{k} v_i \;\wedge\; m_i = \breve{\pi};v_i\,;\rho \\ \Rightarrow Ex - Search(P, m_1, \ldots, m_k, G, S)\ .$$

In Appendix B we show that the following theorem holds:

THEOREM 5.2

$$P = Hd^{\smile} \cdot 1;G;Lst^{\smile} \cdot (Conc^{\smile}\,;(Lst\otimes Hd) \;\rightarrow\; 1;valid)^{\smile}$$
$$\wedge\;\; valid = \sum_{i=1}^{k} v_i \;\wedge\; m_i = \breve{\pi};v_i;\rho \;\wedge\; Ex-Search(P, m_1, \ldots, m_k, G, S)$$
$$\Rightarrow \exists Q\,(P \geq Q \wedge Backtracking(Q, m_1, f_1, \ldots, m_k, f_k, G, S))\;\;.$$

Notice that the relation $Q$ whose existence is guaranteed by the previous theorem does not need to be a refinement of relation $P$ (backtracking does not refine exhaustive search in general), thus the reader may wonder if the relation $Q$ is not smaller than it might be. An analysis of the proof of the theorem shows that this is directly dependent of the election of the feasibility filters $f_1, \ldots, f_k$.

# 6 Conclusions

In this paper we have extended the program design strategies formalized in previous papers to include exhaustive search and backtracking. Also, from a very declarative and generic specification of a class of problems we derived first an exhaustive search solution, which was optimized to a backtracking solution.

# References


[1] Baum G.A., Frias M.F., Haeberer A.M. and Martinez–Lopez P., *From Specifications to Programs: A Fork Algebraic Approach to Bridge the Gap*, in Proceedings of Mathematical Foundations of Computer Science, LNCS 1113, Springer-Verlag, Cracow, Poland, 1996, pp. 180–191.

[2] Frias M.F. and Baum G.A., *On the Exact Expressiveness and Provability of Fork Algebras*, in Abstracts of the Tenth Latinamerican Symposium on Mathematical Logic, Colombia, 1995.

[3] Frias M.F., Baum G.A. and Haeberer A.M., *Adding Design Strategies to Fork Algebras*, in proceedings of Perspective on System Informatics, LNCS 1181, Springer-Verlag, Novosibirsk, Russia, 1996, pp. 214–226.

[4] Frias M.F., Baum G.A. and Haeberer A.M., *Fork Algebras in Algebra, Logic and Computer Science*, Fundamenta Informaticae vol. 32 (1997), pp. 1–25.

[5] Frias, M.F., Baum, G.A. and Haeberer A.M., *A Calculus for Program Construction Based on Fork Algebras, Design Strategies and Generic Algorithms*, to appear in Proceedings of the Fourth Seminar on Relational Methods in Computer Science, Warsaw, Poland.

[6] Frias M.F., Haeberer A.M. and Veloso P.A.S., *A Finite Axiomatization for Fork Algebras*, in Journal of the IGPL, Vol.5, No.3, pp.311–319, 1997.

[7] Haeberer, A.M., Baum, G.A. and Schmidt G., *On the Smooth Calculation of Relational Recursive Expressions out of First–Order Non–Constructive Specifications Involving Quantifiers*, in Proceedings of the International Conference on Formal Methods in Programming and Their Applications, LNCS 735, Springer–Verlag, 1993, 281–298.

[8] Haeberer A.M., Frias M.F., Baum G.A. and Veloso P.A.S., *Chapter on Fork Algebras*, Relational Methods in Computer Science, Springer–Verlag, pp. 54–69, 1997.

[9] Haeberer, A.M. and Veloso, P.A.S., *Partial Relations for Program Derivation: Adequacy, Inevitability and Expressiveness*, in Constructing Programs from Specifications – Proceedings of the IFIP TC2 Working Conference on Constructing Programs from Specifications. North Holland., IFIP WG. 2.1, 1991, 319–371.

[10] Smith, D.R., *Top-Down Synthesis of Divide-and-Conquer Algorithms*, Artificial Intelligence, Vol. 27, 43–96, 1985.

[11] Veloso, P.A.S. and Haeberer, A.M., *A Finitary Relational Algebra for Classical First-Order Logic*, Bulletin of the Section of Logic, University of Lódź, Vol. 20, N. 2, 1991, pp. 52–62.

# A Derivation of the Exhaustive Search Algorithm

In this section we present the derivation of an exhaustive search solution for the problem $P$ specified by

$$P = Hd^{\smile} \cdot 1;G;Lst^{\smile} \cdot (Conc^{\smile};(Lst \otimes Hd) \rightarrow 1;valid)^{\smile} . \tag{7}$$

Notice that the relation $Conc^{\smile};(Lst \otimes Hd)$ is computed by the relation $Pairs$ defined by

$$Pairs = pair + Tl;Pairs,$$

where the relation $pair$ is defined as $Hd \nabla Snd$.

Let us define $g := L^{>0};(Pairs \rightarrow 1;valid)$. By Lemma 3.1 and typing information, we obtain

$$g = L^1;1 + (pair;valid;1) \cdot Tl;g . \tag{8}$$

Let $h := Lst;G;1 \cdot (Pairs \rightarrow 1;valid)$. Then,

$$\begin{array}{cl}
& h \\
= & \{ \text{ by def. } g \text{ and } (8) \ \} \\
& Lst;G;1 \cdot \left(L^1;1 + (pair;valid;1) \cdot Tl;g\right) \\
= & \{ \text{ Boolean algebra } \} \\
& Lst;G;1 \cdot L^1;1 + Lst;G;1 \cdot pair;valid;1 \cdot Tl;g \\
= & \{ \text{ by list and filter promotion } \} \\
& L^1;Lst;G;1 + Tl;Lst;G;1 \cdot pair;valid;1 \cdot Tl;g \\
= & \{ \text{ by list and function promotion } \} \\
& unwrap;G;1 + pair;valid;1 \cdot Tl;(Lst;G;1 \cdot g) \\
= & \{ \text{ fold } h \ \} \\
& unwrap;G;1 + pair;valid;1 \cdot Tl;h .
\end{array}$$

For the next derivation we will use the property

$$1;valid = 1;(1'\nabla m)\ . \tag{9}$$

$$
\begin{array}{cl}
& P \\
= & \{\quad \text{by (7)} \quad\} \\
& Hd^{\smile}\cdot\breve{h} \\
= & \{\quad \text{unfold } h \quad\} \\
& Hd^{\smile}\cdot(unwrap;G;1 \ + \ pair;valid;1 \cdot Tl;h)^{\smile} \\
= & \{\quad \text{converse and Boolean algebra} \quad\} \\
& Hd^{\smile} \cdot 1;G;wrap \ + \ Hd^{\smile} \cdot 1;valid;pair^{\smile} \cdot \breve{h};Tl^{\smile} \\
= & \{\quad \text{by (9) and list} \quad\} \\
& Hd^{\smile} \cdot 1;G;wrap \ + \ Hd^{\smile} \cdot 1;(1'\nabla m);(Hd\nabla Snd)^{\smile} \cdot \breve{h};Tl^{\smile} \\
= & \{\quad \text{by Ax. 9} \quad\} \\
& Hd^{\smile} \cdot 1;G;wrap \ + \ Hd^{\smile} \cdot 1;(Hd^{\smile} \cdot m;Snd^{\smile}) \cdot \breve{h};Tl^{\smile} \\
= & \{\quad \text{by list, } Hd \text{ functional and function promotion} \quad\} \\
& G;wrap \ + \ Hd^{\smile} \cdot m;Snd^{\smile} \cdot \breve{h};Tl^{\smile} \\
= & \{\quad \text{by } m=\sum_{i=1}^{k} m_i\text{, with } m_i \text{ functional} \quad\} \\
& G;wrap \ + \ \sum_{i=1}^{k} Hd^{\smile} \cdot m_i;Snd^{\smile} \cdot \breve{h};Tl^{\smile} \\
= & \{\quad \text{by list, } \breve{h} \text{ left-ideal and function promotion} \quad\} \\
& G;wrap \ + \ \sum_{i=1}^{k} Hd^{\smile} \cdot m_i;Hd^{\smile};Tl^{\smile} \cdot m_i;\breve{h};Tl^{\smile} \\
= & \{\quad \text{by function promotion} \quad\} \\
& G;wrap \ + \ \sum_{i=1}^{k} Hd^{\smile} \cdot m_i;\left(Hd^{\smile}\cdot\breve{h}\right);Tl^{\smile} \\
= & \{\quad \text{fold } P \quad\} \\
& G;wrap \ + \ \sum_{i=1}^{k} Hd^{\smile} \cdot m_i;P;Tl^{\smile} \\
= & \{\quad \text{by list and Ax. 9} \quad\} \\
& G;wrap \ + \ \sum_{i=1}^{k} (1' \ \nabla \ m_i;P);Cons \\
= & \{\quad \text{by } Dom\,(m_i) \leq \neg G \text{ and filter promotion} \quad\} \\
& G;wrap \ + \ \neg G;\sum_{i=1}^{k} (1' \ \nabla \ m_i;P);Cons\ .
\end{array}
$$

# B From Exhaustive Search to Backtracking

In this section we give a sketch of the proof of the following theorem, for the case when $k$ equals 2. The proof can be generalized to arbitrary values of $k$, but we prefer to keep notation simple in order to gain in the understanding of the proof.

THEOREM B.1

$$P = Hd^{\smile} \ \cdot 1;G;Lst^{\smile} \ \cdot (Conc^{\smile} \ ; (Lst \otimes Hd) \ \rightarrow \ 1;valid)^{\smile}$$

$$\wedge \ valid = \sum_{i=1}^{k} v_i \ \wedge \ m_i = \breve{\pi};v_i;\rho \ \wedge \ Ex-Search(P, m_1, \ldots, m_k, G, S)$$

$$\Rightarrow \exists Q \, (P \geq Q \wedge Backtracking(Q, m_1, f_1, \ldots, m_k, f_k, G)) \, .$$

**Sketch of the proof for k = 2** Let us define a relation $ES'$ by the condition

$$ES' = Dom\,(Lst;G) \quad + \quad \neg Dom\,(Lst;G) \, ; \sum_{i=1,2} \begin{pmatrix} Lst;m_i \\ \nabla \\ 1' \end{pmatrix} ; App;ES' \ .$$

Relation $ES'$ is an embedding on the domain of $ES$ from states to sequences of states, and relation $ES$ can be retrieved from $ES'$ as follows: $ES = wrap;ES'$.

In order to prove the theorem it suffices to show that $B_2';Dom\,(Lst;G) \leq ES'$, since then we can reason as follows:

$$\begin{array}{ll} & B_2 \\ = & \{ \ \text{def. } B_2 \ \} \\ & wrap;B_2';Dom\,(Lst;G) \\ \leq & \{ \ \text{hypothesis and monotonicity} \ \} \\ & wrap;ES' \\ = & \{ \ \text{def. } ES \ \} \\ & ES \ . \end{array}$$

The proof proceeds by computational induction. Let $\mathsf{F}$ be the functional associated to the recursive definition of $B_2'$. Let us show that for all $n \in \mathbb{N}$, $\mathsf{F}^n\,(0)\,;Dom\,(Lst;G) \leq ES'$. For $n = 1$ we have:

$$\begin{array}{ll} & \mathsf{F}^1\,(0)\,;Dom\,(Lst;G) \\ = & \{ \ \text{by replacements and simplifications} \ \} \\ & \mathit{2};Dom\,(\rho;Lst;G)\,;\rho;Dom\,(Lst;G) \\ \leq & \{ \ \text{by } Dom \ \} \\ & \mathit{2};\rho;Dom\,(Lst;G) \\ = & \{ \ \text{by fork} \ \} \\ & Dom\,(Lst;G) \\ \leq & \{ \ \text{by def. } ES' \ \} \\ & ES' \ . \end{array}$$

Let us assume that $\mathsf{F}^k\,(0)\,;Dom\,(Lst;G) \leq ES'$. If we unfold the definitions of $M_i$ and $F_i$ $(i = 1, 2)$ in $\mathsf{F}^{k+1}\,(0)$ and distribute compositions over additions as much as possible, then we obtain a sum of terms without additions inside. Among these terms, the following is the most interesting one, being the remaining ones easier to deal with.

$$T := 2;\neg Dom\,(\rho;Lst;G)\,;Dom\,(\pi;f_1)\,;\pi;\left(\begin{array}{c} 1\text{'} \\ \nabla \\ (Lst;m_1\ \nabla\ 1\text{'})\,;App;\mathsf{F}^k\,(0) \end{array}\right)$$
$$;Dom\,(\pi;f_2)\,;\pi;\left(\begin{array}{c} 1\text{'} \\ \nabla \\ (Lst;m_2\ \nabla\ 1\text{'})\,;App;\mathsf{F}^k\,(0) \end{array}\right);\rho;Dom\,(Lst;G)\ .$$

Notice first that

$$\begin{array}{cc}
 & 2;\neg Dom\,(\rho;Lst;G) \\
= & \{\ \text{by } Dom\ \} \\
 & 2;\left(\overline{\rho;Lst;G;1}\cdot 1\text{'}\right) \\
= & \{\ \text{by function promotion twice}\ \} \\
 & \overline{2;\rho;Lst;G;1}\cdot 2 \\
= & \{\ \text{by fork}\ \} \\
 & \overline{Lst;G;1}\cdot 2 \\
= & \{\ \text{by } Dom\ \} \\
 & \neg Dom\,(Lst;G)\,;1\ \cdot\ 2 \\
= & \{\ \text{by filter promotion}\ \} \\
 & \neg Dom\,(Lst;G)\,;2\ .
\end{array}$$

Then,

$$2;\neg Dom\,(\rho;Lst;G) = \neg Dom\,(Lst;G)\,;2\ . \tag{10}$$

$$\begin{array}{cc}
 & T \\
\leq & \{\ \text{by fork and monotonicity}\ \} \\
 & 2;\neg Dom\,(\rho;Lst;G)\,;\pi;(Lst;m_2\ \nabla\ 1\text{'})\,;App;\mathsf{F}^k\,(0)\,;Dom\,(Lst;G) \\
= & \{\ \text{by (10)}\ \} \\
 & \neg Dom\,(Lst;G)\,;2;\pi;(Lst;m_2\ \nabla\ 1\text{'})\,;App;\mathsf{F}^k\,(0)\,;Dom\,(Lst;G) \\
= & \{\ \text{by fork}\ \} \\
 & \neg Dom\,(Lst;G)\,;(Lst;m_2\ \nabla\ 1\text{'})\,;App;\mathsf{F}^k\,(0)\,;Dom\,(Lst;G) \\
\leq & \{\ \text{by inductive hypothesis}\ \} \\
 & \neg Dom\,(Lst;G)\,;(Lst;m_2\ \nabla\ 1\text{'})\,;App;ES' \\
\leq & \{\ \text{by def. } ES'\ \} \\
 & ES'\ .
\end{array}$$