Journal on Relational Methods in Computer Science, Vol. 1, 2004, pp. 181 - 216

Fork Algebras: Past, Present and Future

Marcelo F. Frias*[1], Paulo A. S. Veloso[2], and Gabriel A. Baum[3]

[1] Department of Computer Science, School of Exact and Natural Sciences, Universidad de Buenos Aires, Pabellón I, Ciudad Universitaria, Buenos Aires, 1428, Argentina, and CONICET.
E–mail: mfrias@dc.uba.ar.

[2] Program of Systems and Computer Engineering, COPPE, Universidade Federal do Rio de Janeiro, Caixa Postal 68511, 21945–970 Rio de Janeiro, RJ, Brasil, CNPq and FAPERJ.
E–mail: veloso@cos.ufrj.br

[3] LIFIA, School of Informatics, Universidad Nacional de La Plata, Argentina, and CONICET.
E–mail: gbaum@sol.info.unlp.edu.ar.

Abstract. Fork algebras have interesting connections with Computing, Algebra and Logic. This paper presents a survey of ideas and results about fork algebras, with special emphasis on current developments and promising research lines.

1 Introduction

Fork algebras arose from programming considerations, aiming at a wide-spectrum calculus for program derivation.

Fork algebras are a class of algebras obtained by expanding relation algebras [13, 42, 57] with a new operator called *fork*. Relation algebras are a class of structures axiomatized by a finite set RAE of equations (thus a finitely based variety). These equations axiomatize the behavior of operations $+$, $\cdot$, $^{-}$, 0, 1, ;, $^{\smile}$ and 1', giving properties of structures whose domain is a set of binary relations (on some base set A) closed under *union*, *intersection*, *complement* (relative to $A \times A$), the *empty* binary relation, the *universal* relation $A \times A$, *composition*, *transposition*, and the *identity* relation, respectively. These structures are called *algebras of binary relations*. Composition and transposition are defined set-theoretically as follows:

$$R;S = \{ \langle a, b\rangle : (\exists c \in A)(\langle a, c\rangle \in R \ \wedge \ \langle c, b\rangle \in S) \}, \tag{1}$$

Received by the editors April 16, 2004, and, in revised form, December 02, 2004.
Published on December 10, 2004.

* Research partially supported by a grant from Antorchas Foundation, and Projects UBACyT X094 and PICT 11-05272.

$$\breve{R} = \{ \langle b, a \rangle : \langle a, b \rangle \in R \} \ . \tag{2}$$

If the base set A is closed under an injective binary function $\star$, we can then define the operator *fork* (denoted by ∇) as follow:

$$R \nabla S = \{ \langle a, b \star c \rangle : \langle a, b \rangle \in R \ \wedge \ \langle a, c \rangle \in S \} \ . \tag{3}$$

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

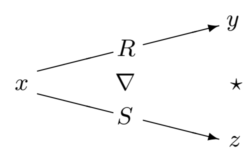

Fig. 1. Fork of binary relations R and S.

The paper is conceptually divided in three parts dealing with previous, current and future research, and assumes from the reader a nodding acquaintance with fundamentals of logic and algebra as can be found in [13].

Previous Research. We will present results showing that the expressive power of fork algebras encompasses that of different classical and non-classical logics. This is very useful in software specification where different logics can be used to specify different views of systems: these specifications can later be mapped to relational specifications in the language of fork algebras. We will also present results showing that every every structure satisfying the axiomatization of fork algebras is isomorphic to a standard model, where the operations have their intended standard meaning. This is known as *representability.* Representability is important in that it implies the completeness of the axiomatization. Therefore, valid properties that follow from a specification can be proved syntactically within the equational calculus of fork algebras. We will finally analyze the suitability of fork algebras as a wide-spectrum programming calculus.

Current Research. One way of giving semantics to diagrammatic notations is by first assigning semantics to diagrams in the most natural logics for each kind of diagram, and then translating these logical specifications to a (uniform) relational algebraic framework. This enterprise sets two demands to be met. On the one hand, one must define what are the logics required to assign formal semantics to diagrams. On the other hand, one must interpret these logics to the relational setting. We will show how to interpret first-order linear temporal logics [43] in an extension of the fork calculus. We are also using fork algebras in order to provide a complete proof calculus for the specification language Alloy [38]. Another line being pursued involves modal versions. Modal logics are convenient formalisms for several purposes, modalities being simpler than quantifiers, but generally have limited expressive power [12]. Arrow logic, for instance, is a modal logic intended to deal with arrows (e.g., transitions) and can handle sequential composition, but not parallelism. The addition of fork to arrow logic provides an extension where one can deal with parallelism and synchronization [5, 6, 20].

Future Research. We will present some preliminary results on the automatic verification of relational specifications. We will also discuss some logics related to fork algebras and how to interpret several logics so that their translation allows to reason across the different logics.

This paper is organized as follows. In Section 2 we briefly describe the evolution of fork algebras. Across Section 3 we present some results about fork algebras obtained in the last few years, including results on the expressiveness and finite axiomatizability of fork algebras, as well as their applications in program development and software specification. In Section 4 we examine current research being done, including work around the **Ar**g**e**ntum project, as well as some other aspects of fork algebras, such as fork modal logics, and their motivations. Finally, in Section 5 we present our conclusions and discuss different open research directions including verification of relational specifications, interpretablity of applied logics, relational semantics for component based software, as well as some other prospects for algebras and logics related to fork algebras and their possible applications.

2 On the Origin of Fork Algebras

Fork algebras have arisen with the goal of providing the foundation of a framework for software specification, verification and derivation.[1]

In our view, specification languages — such as modern graphical notations like UML [11] — must allow for a modular description of the various aspects that comprise a system. These aspects include structural, dynamic and temporal properties. Distinct formalisms allow us to specify each one of these aspects, namely, first-order classical logic for structural properties, propositional and first-order

[1] In fact, fork algebras have originated from an algebraic calculus of problems and the attempts to apply these ideas to program construction [19]. Some intuitive ideas of Pólya [53] were given precise formulations using the idea of a problem as an input-output relation [59]. Operations on problems allow the formulation of some problem-solving methods, but divide-and-conquer involves parallel actions.

dynamic logic for dynamic properties, and different temporal logics for temporal properties. Some of the previously mentioned formalisms have complete deductive systems. Nevertheless, reasoning across formalisms may be rather difficult, if not impossible. A possible approach to overcome this problem amounts to finding a suitable amalgamating formalism: one that is expressive enough to interpret the specification formalisms, with a simple semantics understandable by non-mathematicians, and with a complete and simple deductive system. If by "simple deductive system" we mean a complete equational calculus with finitely many axioms, then the algebras of binary relations are not an adequate candidate, because they cannot be axiomatized with a finite number of equations [46]. Fortunately, as we will see in Section 3.2, fork algebras are a perfect candidate.

Notice that the definition of fork strongly depends on the function $\star$. Actually, the definition of fork evolved *around* the definition of the function $\star$. From 1990 (when the first class of fork algebras was introduced) until now, different alternatives were explored with the aim of finding a framework which would satisfy our needs. In the definition of the first class of fork algebras by Veloso and Haeberer [65], function $\star$ produced true set theoretical pairs, i.e., when applied to values a and b, $\star(a, b)$ returned the pair $\langle a, b\rangle$. Mikulás, Sain, Simon and Németi showed in [45, 54] that this class of fork algebras was not finitely axiomatizable.[2] Other classes of fork algebras were defined, in which $\star$ was binary tree formation or even concatenation of sequences, but these were shown to be non finitely axiomatizable too. It was in [28] that the class of fork algebras to be used in this paper came up. The only requirement placed on function $\star$ was that it must be injective. This was enough to prove in [29] that the newly defined class of fork algebras was indeed axiomatized by a finite set of simple equations. This is the axiomatization adopted nowadays.

3 The Development of Fork Algebras

In this section we analyze previous results about fork algebras. In Section 3.1 we present the classes of proper and abstract fork algebras, as well as some extensions. In Section 3.2 we show that the axioms for fork algebras presented in Section 3.1 indeed fully characterize the class of proper fork algebras. In Section 3.3 we show that there are no superfluous axioms in the axiomatization proposed. In Section 3.4 we analyze the expressive power of fork algebra terms and equations. In Section 3.5 we present results on the interpretability of different modal and multimodal logics.

[2] This was done by proving that a sufficiently complex theory of natural numbers can be interpreted in the equational theory of these fork algebras, and thus leads to a non recursively enumerable equational theory.

3.1 Proper and Abstract Fork Algebras

It is a standard procedure in algebra to define classes of algebras by operating on some previously defined classes. For instance, Boolean algebras can be defined as the closure of the class of *Set Boolean Algebras* (those Boolean Algebras whose elements are sets and the operations are the complement of a set, union of two sets, etc.) under subalgebras, direct products and homomorphic images. We will follow a similar procedure in order to define the class of proper fork algebras (denoted by PFA). To define the class PFA, we will first define the class of *Pre Proper Fork Algebras*, denoted by $\star$PFA.

First, given a relation $\star \subseteq (U \times U) \times U$, we can use it to define a binary operation on binary relations on the set U, namely, the *induced fork operation* $\underline{\star}$, defined by:

$$S\underline{\star}T = \{\, \langle x, z\rangle \in U \times U : \exists z', z'' \in U \text{ s.t. } x\,S\,z' \;\wedge\; x\,T\,z'' \;\wedge\; \langle z', z''\rangle \star z \,\}\,.$$

A graphical interpretation of the fork operation $\underline{\star}$ induced by a function[3] $\star : U \times U \to U$ is given in Fig. 2.

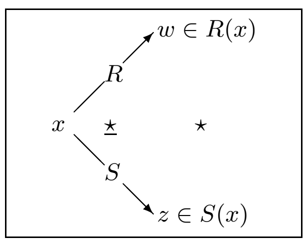

Fig. 2. The operator fork.

We can now define the pre proper fork algebras as two-sorted structures.

Definition 1. *A* pre proper fork algebra *on the domains P and U is a (two-sorted) algebraic structure $\langle\, P, U, \cup, \cap, ^-, \emptyset, V, \circ, Id, ^{\smile}\,, \nabla, \star\rangle$ such that:*

1. *$\langle\, P, \cup, \cap, ^-, \emptyset, V, \circ, Id, ^{\smile}\,\rangle$ is an algebra of binary relations on the set U,*
2. *$\star : U \times U \to U$ is a binary function that is injective on the restriction of its domain to V,*
3. *P is closed under the fork operation ∇ induced by $\star$,*

Notice that, in Def. 1 (2), V is a binary relation on U, and thus the restriction of $U \times U$ to V is adequate. Proper fork algebras are obtained as reducts (by forgetting some operations and domains) of pre proper fork algebras.

[3] When $\star$ is a function, we have $S\underline{\star}T = \{\, \langle x, \star(z', z")\rangle \in U \times U : x\,S\,z' \;\wedge\; x\,T\,z" \,\}$.

Definition 2. *We define the class of proper fork algebras as* $\mathbf{Rd}\star\mathsf{PFA}$*, where* $\mathbf{Rd}$ *takes reducts to the similarity type* $\langle \cup, \cap, {}^{-}, \emptyset, V, \circ, Id, {}^{\smile}, \nabla \rangle$.

Notice that proper fork algebras are obtained from pre proper fork algebras by forgetting the domain U and the function $\star$. The function $\star$ performs the role of pairing, encoding pairs of objects into single objects. It is important to bear in mind that there are $\star$ functions which are distinct from set-theoretical pair formation: $\star(x, y)$ differs from the ordered pair $\langle x, y\rangle$ (i.e., $\{ x, \{ x, y \} \}$).

Simple examples of these algebras are the *square pre proper fork algebras* and *square proper fork algebras*: those with universal relation $V = U \times U$ and $U \neq \emptyset$.

Given a PFA $\mathfrak{A}$ with base $U_{\mathfrak{A}}$, it is possible to single out those elements (if any) that do not represent pairs. Notice that the term $\overline{1\nabla 1}$ stands for the binary relation $\{ \langle x, y\rangle \in U_{\mathfrak{A}} : \forall u, v \in U_{\mathfrak{A}} \,(y \neq \star(u, v)) \}$. Thus, the term $Ran\left(\overline{1\nabla 1}\right)$ characterizes those elements in the base that are not pairs. In what follows we will denote by 1'_{U} the term $Ran\left(\overline{1\nabla 1}\right)$, and by ${}_{\mathsf{U}}1$ the term $1\text{'}_{\mathsf{U}};1$. We will call the elements from the base in the domain of 1'_{U} *urelements*, and will denote the set of urelements of a fork algebra $\mathfrak{A}$ by $Urel_{\mathfrak{A}}$. Under the previous definitions, the equation

$$1;1\text{'}_{\mathsf{U}};1 = 1 \tag{4}$$

is valid in a proper fork algebra $\mathfrak{A}$ only in case $Urel_{\mathfrak{A}}$ is nonempty.

Given a pair of binary relations, the operation called *cross* (and denoted by $\otimes$) performs a kind of parallel product. A graphic representation of cross is given in Fig. 3. Its set theoretical definition is given by

$$R \otimes S = \{ \langle \star(x, y), \star(w, z)\rangle : x\,R\,w \wedge y\,S\,z \}\,.$$

$$\boxed{\begin{array}{ccc} x \longrightarrow & R & \longrightarrow w \in R(x) \\ \star & \otimes & \star \\ y \longrightarrow & S & \longrightarrow z \in S(y) \end{array}}$$

Fig. 3. The operator cross.

It is not difficult to check that cross is definable from the other relational operators with the use of fork. It is a simple exercise to show that if V is the greatest relation in a proper fork algebra, then

$$R \otimes S = ((Id\nabla V)^{\smile} \circ R)\nabla((V\nabla Id)^{\smile} \circ S).$$

Much as relation algebras are abstract versions of algebras of binary relations, proper fork algebras also have their abstract counterparts, forming the class of abstract fork algebras (AFA).

Definition 3. *An* abstract fork algebra *is an algebraic structure*

$$\langle\, A, +, \cdot, ^{-}, 0, 1, ;, 1', ^{\smile}, \nabla \,\rangle,$$

where

1. *the reduct* $\langle\, A, +, \cdot, ^{-}, 0, 1, ;, 1', ^{\smile} \rangle$ *is a relation algebra,*
2. *for all* $r, s, t, q \in A$*,*

$$r \nabla s = (r; (1' \nabla 1)) \cdot (s; (1 \nabla 1')) \,, \qquad \textit{(Ax. 1)}$$

$$(r \nabla s) \,;^{\smile} t \nabla q = (r;^{\smile} t) \cdot (s;^{\smile} q) \,, \qquad \textit{(Ax. 2)}$$

$$(1' \nabla 1)^{\smile} \nabla (1 \nabla 1')^{\smile} \leq 1'. \qquad \textit{(Ax. 3)}$$

Thus, the abstract fork algebras are axiomatized by the set FAE extending the set RAE of relation-algebra equations by the above three fork axioms.

From the abstract definition of fork induced by the axioms in Def. 3, it is possible to define cross by the equation

$$R \otimes S = ((1' \nabla 1)^{\smile}; R) \nabla ((1 \nabla 1')^{\smile}; S) \ . \qquad (5)$$

The terms $(1' \nabla 1)^{\smile}$ and $(1 \nabla 1')^{\smile}$ deserve special attention, we call them π and ρ respectively, and they are quasi-projections.[4] When interpreted in a proper fork algebra, these terms behave as projections, projecting components from pairs constructed with an injective function $\star$. Figure 4 illustrates the meaning of these relations. They will allow us to cope in further sections with the lack of variables over individuals in the language of abstract fork algebras.

Notice that under the previous definitions of π and ρ, (5) can be spelled in a simpler form as follows:

$$R \otimes S = (\pi; R) \nabla (\rho; S) \ .$$

Also, the above axioms Ax. 1 and Ax. 3 can be written as follows:

$$r \nabla s = (r; \breve{\pi}) \cdot (s; \breve{\rho}) \,, \qquad (6)$$

[4] We see that the axiom Ax. 2 yields functionality ($\breve{\pi};\pi \leq 1'$ and $\breve{\rho};\rho \leq 1'$) and $\breve{\pi};\rho = 1$, whereas axioms Ax. 1 and Ax. 3 yield joint injectivity $((\pi;\breve{\pi}) \cdot (\rho;\breve{\rho}) \leq 1')$.

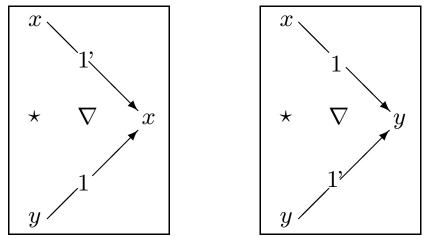

Fig. 4. The projections π and ρ.

$$\pi \nabla \rho \leq 1' \ . \tag{7}$$

Abstract fork algebra arithmetic is quite rich. As examples of derived properties, we mention monotonicity of ∇ and $\otimes$, annihilation of ∇ and $\otimes$ by 0, and distributions

$$(r+s)\nabla t = (r\nabla t)+(s\nabla t), \tag{8}$$

$$(r\nabla s)\cdot(t\nabla q) = (r\cdot t)\nabla(s\cdot q), \tag{9}$$

$$(r\nabla s);(t\otimes q) = (r;t)\nabla(s;q), \tag{10}$$

$$(r\otimes s)^{\smile} = \breve{r}\otimes\breve{s} \ . \tag{11}$$

as well as interdefinabilities

$$r\nabla s = \mathit{2};(r\otimes s), \tag{12}$$

$$r\cdot s = (r\nabla s);\breve{\mathit{2}}, \tag{13}$$

where $\mathit{2} = 1'\nabla 1'$ is the duplication constant.

Also, the algebraic structure of abstract fork algebras is parallel to that of relation algebras [39]: an abstract fork algebra $\mathfrak{A}$ is simple iff its relational reduct $\mathfrak{A}_r$ is simple (a simple algebra is one with no proper homomorphic image[5]) and every abstract fork algebra is isomorphic to a subdirect product of simple homomorphic images[6]. As a consequence, a fork algebra equation holds in all abstract fork algebras iff it holds in all simple fork algebras.

3.2 Finite Axiomatizability

We will now show that the axioms for fork algebras given in Section 3.1 indeed fully characterize the class of proper fork algebras.

[5] The square proper fork algebras are simple.

[6] These properties follow from the following construction (due essentially to axiom (1), as rewritten above): each homomorphic image of the relational reduct of a fork algebra can be expanded to a fork algebra.

We will establish that every abstract fork algebra is isomorphic to a proper fork algebra, by relying on the representability of abstract relation algebras with quasi-projections by proper algebras of binary relations [58] as well as on a simple construction of pairing based on pair of relations.

Given an abstract fork algebra $\mathfrak{A} = \langle A, +, \cdot, ^-, 0, 1, ;, 1', ^\smile, \nabla \rangle$, we know that its relational reduct $\mathfrak{A}_r = \langle A, +, \cdot, ^-, 0, 1, ;, 1', ^\smile \rangle$ is a relation algebra where the defined terms π and ρ are quasi-projections (cf. 3.1). The representability of quasiprojective abstract relation algebras yields an isomorphism $h : A \to P$ of the relational reduct $\mathfrak{A}_r$ onto an algebra of binary relations $\mathfrak{P} = \langle P, \cup, \cap, ^-, \emptyset, V, \circ, Id, ^\smile \rangle$ on set U. We now use the relations $h(\pi)$ and $h(\rho)$ in P to define the pairing relation $\star \subseteq (U \times U) \times U$ by $\langle \langle x, y \rangle, z \rangle \in \star$ iff $\langle z, x \rangle \in h(\pi)$ and $\langle z, y \rangle \in h(\rho)$. We can now see that the restriction of the relation $\star$ to V is an injective function (thus the expansion $\mathfrak{P}^{\underline{\star}} = \langle P, \cup, \cap, ^-, \emptyset, V, \circ, Id, ^\smile, \underline{\star} \rangle$ is a PFA) and $h(r \nabla s) = h(r) \underline{\star} h(s)$ (and thus bijection $h : A \to P$ becomes an isomorphism of abstract fork algebra $\mathfrak{A}$ onto the PFA $\mathfrak{P}^{\underline{\star}}$).

We thus have the representability of fork algebras: abstract by proper ones.

Theorem 1. *Every abstract fork algebra is isomorphic to a proper fork algebra.*

This representation theorem has some interesting methodological consequences.

1. On the intuitive side, the objects of abstract fork algebras can be regarded as relations of input-output pairs.
2. On the technical side, proper and abstract fork algebras have the same abstract properties, which yields the completeness of the fork algebra equational axiomatization *FAE* (in Section 3.1) with respect to proper fork algebras. Also, a fork algebra equation holds in all proper fork algebras iff it can be derived from the set *FAE* by equational reasoning.

Theorem 2. *Let $E \cup \{e\}$ be a set of fork algebra equations. Then,*

$$E \models_{\mathsf{PFA}} e \quad \Longleftrightarrow \quad E \vdash_{FAE} e\ .$$

3.3 Independence of the Axioms

If we use the language of fork algebras as a specification language, then avoiding superfluous axioms helps keeping this formalism easier to handle. In [24, 62] we proved that all the axioms are independent, that is, removing any axiom yields a strictly larger class of algebras.

3.4 Expressiveness of the Fork Algebra Terms and Equations

We have seen in Section 3.2 the representability theorem stating that $\mathsf{AFA} = I[\mathsf{PFA}]$. As a consequence, proper and abstract fork algebras have the same properties (expressible in their common abstract language). The caveat is "expressible in the abstract language of AFA". In this language we treat relations as opaque objects: without considering their input-output pairs or their individual components. This has some methodological advantages for program derivation. But, are we not losing expressive power? For reasoning (about programs and specifications), it is important to have adequate power for expressing their properties. The expressiveness results will show that the fork algebra terms can express first-order properties: for each first-order formula we can effectively construct a closed (variable-free) fork algebra term with the same extension. Moreover, first-order derivations correspond to equational derivations from the fork algebra equational axiomatization *FAE* (in Section 3.1).

We will now analyze the expressive power of fork algebras. We will show that properties expressed by fist-order formulas can also be expressed with fork algebraic terms and that fork algebra equations can express all first-order sentences.

To convert fist-order formulas to fork algebraic terms, we have the Boolean apparatus for the sentential connectives and we can view the existential quantifier as a search, which can be expressed by composite projection terms [33, 65].

An appropriate framework for these considerations is the so-called elementary theory of (simple) fork relations. It is based on the idea of describing relations by means of their extensions, much in the spirit of Tarski's elementary theory of relations [57]. Recall that in a square pre proper fork algebra, we can describe each operation on binary relations by its set of input-output pairs (cf. (1), (2) and (3) in Section 1).

Definition 4. *The* elementary theory of simple fork relations *(SFR, for short) is the two-sorted theory*

- *with similarity type Σ_E consisting of three subtypes*
  - *Σ_U, consisting of a sort U and an operation $\star : U \times U \to U$,*
  - *Σ_R, consisting of a sort R and the fork algebra operations on sort R,*
  - *$\Sigma_{[\ ]}$, consisting of sorts U and R and a ternary relation $[\ ]$ ($u\ [\ r\ ]\ z$ is intended to mean that the pair $\langle u, z\rangle$ of U-objects belongs to the R-object r)*
- *and axioms*
  - *stating that the operation $\star : U \times U \to U$ is injective,*
  - *defining the symbols in $\langle +, \cdot, {}^{-}, 0, 1, ;, 1', {}^{\smile}, \nabla \rangle$ by their $[\ ]$-extensions, much as (1), (2) and (3) in Section 1. For instance, $\forall u \forall z u[1]z$ and $u[1']z \leftrightarrow u = z$ define 1 (as $U \times U$) and $1'$ (as Id).*

- *extensionality expressing that R-objects with the same [ ]-extension are equal.*

Each square pre proper fork algebra is a model of *SFR* (where [ ] is $\in$). Actually, in view of extensionality, these are the only models of *SFR*, up to isomorphism. Also, the theory $SFR \subseteq L_E$ is a common conservative extension[7] of the following two theories:

– the theory $BIO \subseteq L_U$ of binary injective operation $\star : U \times U \to U$,
– the theory $Th(\mathsf{PFA}_s) \subseteq L_R$ of square proper fork algebras, $\star : U \times U \to U$.

We will use this remark later on.[8] We wish to convert each first-order formula to a fork algebraic term, with the same extension. For this purpose, we code the free variables.[9] The *code* of set $\underline{v} := \{v_{i_1}, v_{i_2}, \dots, v_{i_n}\}$ is $K_n(\underline{v}) := (v_{i_1} \star (v_{i_2} \dots \star v_{i_n}) \dots)$.[10] We thus have fork algebraic terms to extract

– a variable v_{i_j}: $e(\underline{v}, v_{i_j})$ (composed of projections),
– the code of a subset $\underline{u} \subseteq \underline{v}$: $E(\underline{v}, \underline{u})$ (consisting of projections and ∇'s).

We will convert a first-order formula θ, with set $\underline{w}$ of $m > 0$ free variables, to a fork algebraic term $\mathcal{I}(\theta)$, having extension $K_m(\underline{w})$, i.e.,

$$SFR \vdash K_m(\underline{w}) = z \to (\theta \leftrightarrow z \;[\; \mathcal{I}(\theta) \;]\; z)\ .$$

We define this translation as follows

[=] $\mathcal{I}(u = v) := 2;\breve{2}$,
[$\star$] $\mathcal{I}(w = u \star v) := E;\breve{E}$, with $E := E(\{u, v, w\},\ \{u, v\})$,
[$\neg$] $\mathcal{I}(\neg\theta) := \overline{\mathcal{I}(\theta)} \cdot 1'$,
[$\wedge$] $\mathcal{I}(\psi \wedge \theta) := (E_\psi;\mathcal{I}(\psi);(E_\psi)\breve{}) \cdot (E_\theta;\mathcal{I}(\psi);(E_\theta)\breve{}) \cdot 1'$, where terms E_ψ and E_θ are defined by $E_\psi := E(\underline{w}, \mathit{freevars}(\psi))$ and $E_\theta := E(\underline{w}, \mathit{freevars}(\theta))$,
[$\exists$] $I(\exists u\theta) := (E_\theta)\breve{};I(\theta);E_\theta$, where $E_\theta := E(\mathit{freevars}(\theta), \underline{w})$.[11]

We thus have the expressiveness of fork algebra terms.

---

[7] A theory $\Gamma' \subseteq L'$ is a conservative extension of a theory $\Gamma \subseteq L$ (noted $\frac{\Gamma}{L} \sqsubseteq \frac{\Gamma'}{L'}$ or $\Gamma \sqsubseteq \Gamma'$) when both prove the same sentences of the smaller language ($\Gamma' \vdash \tau$ iff $\Gamma \vdash \tau$, for every sentence of $L \subseteq L'$). A useful sufficient condition for conservativeness is model expandability: every model of the smaller theory $\Gamma \subseteq L$ can be expanded to a model of the larger theory $\Gamma' \subseteq L'$. Extensions by definitions are conservative.

[8] We have $\frac{BIO}{L_U} \sqsubseteq \frac{SFR}{L_E} \sqsupseteq \frac{Th(\mathsf{PFA}_s)}{L_R}$.

[9] The extension of a fork algebraic term is a binary relation, while that of a formula is an n-ary relation.

[10] One may regard this code as an assignment.

[11] This is the case when $u \in \underline{w}$, otherwise, we set $I(\exists u\theta) := I(\theta)$.

Lemma 1. *For every first-order formula θ of L_U, with set $\underline{w}$ of $m > 0$ free variables, $\mathcal{I}(\theta)$ is a closed term of L_R, such that $SFR \vdash K_m(\underline{w}) = z \to (\theta \leftrightarrow z\ [\ \mathcal{I}(\theta)\ ]\ z)$.*

We have translated each first-order formula to a closed fork algebraic term, from which we can recover the extension of the formula. The converse translation is simpler: each closed fork algebraic term t of L_R can be converted to a first-order formula $\eta(t)$ of L_U with two free variables u and z, so that $SFR \vdash u\ [\ t\ ]\ z \leftrightarrow \eta(t)$ (cf. also 4.3 in Section 4).

We still have two points to deal with:

1. The operation $\star : U \times U \to U$ may fail to be part of our repertoire.[12]
2. We usually also have (other) predicates and functions.

We will now examine these two issues, beginning with thefirstone.Given a domain of individuals, we may view it as a sort D and create sort U consisting of the trees with leaves in D, under operation $\star : U \times U \to U$. We thus have an injection $j : D \to U$ mapping D to the urelements of U (denoted by $j[D, Ur(U)]$).[13] We can then translate each formula φ of L_D, with set $\underline{x}$ of m free variables, to a formula φ_U of L_U, with set $\underline{u}$ of m free variables, so that $j[D, Ur(U)] \vdash j(\underline{x}) = \underline{u} \to (\varphi \leftrightarrow \varphi_U)$.[14]

Now, we can compose both translations.

Proposition 1. *For every first-order formula φ of L_D, with set $\underline{x}$ of $m > 0$ free variables, we can construct a closed term $t(\varphi)$ of L_R, such that*

$$SFR \cup j[D, Ur(U)] \vdash K_m j(\underline{x}) = z \to (\varphi \leftrightarrow z\ [\ t(\varphi)\ ]\ z)\ .$$

We have translated each first-order formula (with free variables) to a closed fork algebraic term, from which we can recover the extension of the formula. Let us now extend the translation to first-order sentences. Formula φ is translated to a closed term $t(\varphi)$ on R so that φ is equivalent to $z\ [\ t(\varphi)\ ]\ z$. So, it is natural to translate $\forall x\varphi$ to the equation $t(\varphi) = 1'$. Also, $\exists x\varphi$ is translated to $t(\varphi) \neq 0$, which is equivalent to $1;t(\varphi);1 = 1$, for simple algebras.

We translate each first-order sentence σ to an equation $t_\sigma = 1'$ as follows.

[$\forall$] $t_{\forall x\varphi} := t(\varphi)$,
[$\neg$] $t_{\neg\sigma} := (\ 1;\overline{t_\sigma};1\ )\cdot 1'$,
[$\wedge$] $t_{\sigma\wedge\tau} := t_\sigma \cdot t_\tau$.

[12] We may regard the objects of sort U as structured by $\star$: binary trees, rather than mere individuals.
[13] We have a conservative extension $SFR \sqsubseteq SFR \cup j[D, Ur(U)]$.
[14] The set of urelements is definable (cf. Section 3.1).

We thus have the expressiveness of fork algebra equations.

Lemma 2. *For every first-order sentence σ of L_D, we can construct a closed term t_σ of L_R, such that* $SFR \cup j[D, Ur(U)] \vdash \sigma \leftrightarrow t_\sigma = 1'$.

We can now examine the other issue mentioned before: the repertoires of predicates and functions. We can transfer predicates and functions, using the connections j, from D to U, and [ ], from U to R. For each m-ary predicate p on D, we define an m-ary predicate $p_U := j[p]$ over U, and for each m-ary predicate q on U, we introduce a constant c_q on R. We can define m-ary predicate p_U on U by

$$p_U(\underline{u}) \leftrightarrow (\exists \underline{x} : D)\ [p(\underline{x}) \wedge j(\underline{x}) = \underline{u}]\ .$$

Similarly, for n-ary function f on D we can define n-ary function f_U on U by

$$f_U(\underline{u}) = u \leftrightarrow (\exists \underline{x}, x : D)\ [f(\underline{x}) = x \wedge j(\underline{x}) = \underline{u} \wedge j(x) = u]\ .$$

We can introduce constant c_q on R by

$$u\ [\ c_q\ ]\ v \leftrightarrow u = v \wedge (\exists \underline{u} : U)\ [q(\underline{u}) \wedge K_m(\underline{u}) = u]$$

and similarly for n-ary function g on U we can introduce constant c_g on R by

$$u\ [\ c_g\ ]\ v \leftrightarrow (\exists \underline{u} : U)\ [g(\underline{u}) = v \wedge K_m(\underline{u}) = u]\ .$$

Now, we can compose both transfers to transfer each symbol s on D to a constant $s^* := c_{s_U}$ on R. We can also extend the above translations.

Given repertoires P and F of predicates and functions on D, we can construct corresponding repertoires P^* and F^* of constants on R. We can also extend theory SFR to $SFR^*[P, F]$ by adding the set $Eq[F^*] \subseteq L_R[P^*, F^*]$ of equations expressing the functionality, and arity, of each constant $c^* \in F^*$. We add the translation of $(\forall \underline{x} : D)(\exists! y : D)[f(\underline{x}) = y]$. This construction gives $SFR^*[P, F]$ as a common conservative extension of $SFR \subseteq L_E$ and the theory $Th(\mathsf{PFA}_s) \cup Eq[F^*] \subseteq L_R[P^*, F^*]$. We have

$$\frac{SFR}{L_E} \sqsubseteq SFR^*[P, F] \sqsupseteq \frac{Th(\mathsf{PFA}_s) \cup Eq[F^*]}{L_R[P^*, F^*]}\ .$$

We thus have the overall expressiveness of fork algebra terms and equations.

Proposition 2. *Given repertoires P and F of predicates and functions on D, consider corresponding repertoires P^* and F^* of constants on R.*

1. *For every first-order formula φ of $L_D[P,F]$, with set $\underline{x}$ of $m > 0$ free variables, we can effectively construct a closed term t_φ of $L_R[P^*,F^*]$, such that $SFR^*[P,F] \vdash K_m j(\underline{x}) = z \to (\varphi \leftrightarrow z\ [\ t_\varphi\ ]\ z)$.*
2. *For every first-order sentence τ of $L_D[P,F]$, we can effectively construct a variable-free equation $\varepsilon(\tau)$ of $L_R[P^*,F^*]$, such that $SFR^*[P,F] \vdash \tau \leftrightarrow \varepsilon(\tau)$.*

We can now put together the observations about our constructions and conservativeness. We can see that the theory $SFR^*[P,F]$ is a conservative extension of the set of the validities $Th(\emptyset) \subseteq L_D[P,F]$ and the theory $Th(\mathsf{PFA}_s) \cup Eq[F^*] \subseteq L_R[P^*,F^*]$.[15]

Thus, given a set of sentences $\Sigma \cup \{\tau\}$ of $L_D[P,F]$, we have a set of variable-free equations $\varepsilon(\Sigma) \cup \varepsilon(\tau)\}$ of $L_R[P^*,F^*]$, such that $\Sigma \vdash \tau$ iff $Th(\mathsf{PFA}_s) \cup Eq[F^*] \cup \varepsilon(\Sigma) \vdash \varepsilon(\tau)$.

We thus have the adequacy of fork algebra equations.

Theorem 3. *Given repertoires P and F of predicates and functions on D, consider corresponding repertoires P^* and F^* of constants on R.*

1. *For every set of sentences $\Sigma \cup \{\tau\}$ of $L_D[P,F]$, we have a set of variable-free equations $\varepsilon(\Sigma) \cup \{\varepsilon(\tau)\}$ of $L_R[P^*,F^*]$, such that*
$$\Sigma \vdash \tau \quad \Longleftrightarrow \quad FAE \cup Eq[F^*] \cup \varepsilon(\Sigma) \vdash \varepsilon(\tau)\ .$$
2. *In the case without functions ($F = \emptyset$), for a set of sentences $\Sigma \cup \{\tau\}$ in $L_D[P,\emptyset]$, the set of variable-free equations $\varepsilon(\Sigma) \cup \varepsilon(\tau)$ is in $L_R[P^*,\emptyset]$, and*
$$\Sigma \vdash \tau \quad \Longleftrightarrow FAE \cup \varepsilon(\Sigma) \vdash \varepsilon(\tau)\ .$$

3.5 Interpretability of Non-Classical Logics to Fork Algebras

In Section 3.4 we proved that the expressive power of the fork algebra equation encompasses that of classical first-order logic. In this section we will extend this result to other logics. We will show that specifications written in logics such as propositional or first-order dynamic logic and other modal logics can be fully captured within (extensions of) the calculus of fork algebras. Much the same as in Section 3.4 we defined mapping $\mathcal{I}$ translating classical first-order formulas to fork terms, in this section given a logic $\mathcal{L}$, we will define a mapping $T_\mathcal{L}$ translating formulas from $\mathcal{L}$ to fork algebra terms in such a way that validity of a formula α in $\mathcal{L}$ reduces to proving equationally some equation involving $T_\mathcal{L}(\alpha)$. In the remaining parts of this section we will introduce the logics to be translated as well as the mappings, the interpretability theorems and pointers to their proofs.

[15] We have $\frac{Th(\emptyset)}{L_D[P,F]} \sqsubseteq SFR^*[P,F] \sqsupseteq \frac{Th(\mathsf{PFA}_s)\cup Eq[F^*]}{L_R[P^*,F^*]}$.

Interpretability of Propositional Modal Logics In this section we show that a broad class of propositional modal logics (including the multimodal propositional dynamic logic) can be interpreted in the equational calculus of fork algebras. This interpretability enables us to develop a fork-algebraic formalization of these logics and, as a consequence, to simulate nonclassical means of reasoning with equational theories of fork algebras. The idea of relational formalization of logical systems has been originated in Orlowska ([47]) and developed further in Orlowska ([48–50]). Examples of relational formalisms for applied logics can also be found in Buszkowski and Orlowska ([15]), Demri and Orlowska ([18]), Demri et al. ([17]), Herment and Orlowska ([37]).

The standard semantics of nonclassical logics is usually defined in terms of frames (Kripke [40, 41]), that is, relational systems $\mathfrak{F} = \langle W, R \rangle$ consisting of a set W of states and an accessibility relation R on W. Models are obtained from frames by adding a valuation for the set P of propositional variables. Thus, a model is a triple $\mathfrak{M} = \langle W, R, V \rangle$, where $V : P \to 2^W$. We can extend V to a function $\mathfrak{M}[\alpha]$ that assigns to a formula α the set of states where the formula holds.

The interpretability of a nonclassical logic in fork algebras is established by means of a provability preserving translation of formulas of the logic into fork algebra equations. Under that translation both formulas, formerly understood as sets of states, and accessibility relations receive a uniform representation as relations. The propositional connectives are transformed into relational operations. The constraints on accessibility relations are translated into relational equations. The major advantage of relational formalization is that it provides a uniform framework for representation of a broad class of applied logics and enables us to apply an equational proof theory to these logics.

The inductive definition of satisfiability describes the truth conditions depending on the complexity of formulas. For atomic formulas (propositional variables) we have:

(at) $(\mathfrak{M}, w) \models p$ iff $w \in V(p)$ for any propositional variable p.

For formulas built with extensional operators such as classical negation, disjunction, conjunction or implication, their satisfiability at a possible world is completely determined by satisfiability of their subformulas at that world:

($\neg$) $(\mathfrak{M}, w) \models \neg\alpha$ iff not $(\mathfrak{M}, w) \models \alpha$
($\vee$) $(\mathfrak{M}, w) \models \alpha \vee \beta$ iff $(\mathfrak{M}, w) \models \alpha$ or $(\mathfrak{M}, w) \models \beta$
($\wedge$) $(\mathfrak{M}, w) \models \alpha \wedge \beta$ iff $(\mathfrak{M}, w) \models \alpha$ and $(\mathfrak{M}, w) \models \beta$
($\to$) $(\mathfrak{M}, w) \models \alpha \to \beta$ iff $(\mathfrak{M}, w) \models \neg\alpha \vee \beta$.

For formulas defined from modal operators, as $[R]$ (necessity) and $\langle R\rangle$ (possibility), we have

($[R]$) $(\mathfrak{M}, w) \models [R]\alpha$ iff for all $u \in W, (w,u) \in R$ implies $(\mathfrak{M}, u) \models A$
($\langle R\rangle$) $(\mathfrak{M}, w) \models \langle R\rangle\, \alpha$ iff there is an $u \in W$ s.t. $(w,u) \in R$ and $(\mathfrak{M}, u) \models A$.

For the sake of simplicity, we use the same symbol for the relational constant R that appears in modal operators and the accessibility relation which is denoted by this constant.

Before defining the mapping T_M, we define the mapping T'_M by:

(var) $T'_M(p_i) = P_i$, where p_i is a propositional variable and P_i is a relational variable,
(neg) $T'_M(\neg\alpha) = 1'_{\mathsf{U}};\overline{T'_M(\alpha)}$,
(and) $T'_M(\alpha \wedge \beta) = T'_M(\alpha) \cdot T'_M(\beta)$,
(or) $T'_M(\alpha \vee \beta) = T'_M(\alpha) + T'_M(\beta)$,
($\langle R\rangle$) $T'_M(\langle R\rangle\, \alpha) = R;T'_M(\alpha)$.
($[R]$) $T'_M([R]\alpha) = T'_M(\neg\, \langle R\rangle\, \neg\alpha)$.

For the sake of simplicity we assume that the constant R from the modal language is translated into a constant from the language of fork algebras that is denoted by the same symbol. We finally define the mapping T_M by $T_M(\alpha) = T'_M(\alpha) + \overline{_{\mathsf{U}}1}$.

Under the previous definitions, the following theorem is proved in [30].

Theorem 4. *Given a set of modal formulas $\Psi \cup \{\, \varphi \,\}$,*

$$\Psi \models \varphi \qquad \iff \qquad \{\, T_M(\psi) = 1 : \psi \in \Psi \,\} \vdash_{\mathsf{AFA}} T_M(\varphi) = 1\ .$$

Notice that in the previous theorem there is no mention made about the modal logic being interpreted. If some properties of the accessibility relation are required, then we add these properties as equations to be used along derivations.

Interpretability of Propositional Dynamic Logic Propositional dynamic logic [34] is considered as a programming logic, i.e., a logic suitable for asserting and proving properties of programs. Dynamic logic is a modal logic whose modal operators are determined by programs understood as binary relations in a set of computation states. The following definitions provide a formal description of propositional dynamic logic.

Definition 5. *Let us consider a set P_0 of atomic programs, and a set F_0 of atomic dynamic formulas. From these sets we construct the sets F of dynamic formulas and P of compound programs. F and P are the smallest sets satisfying the following conditions:*

1. *true* $\in F$; *false* $\in F$; $F_0 \subseteq F$,
2. *if* $p \in F$ *and* $q \in F$ *then* $\neg p \in F$ *and* $(p \vee q) \in F$,
3. *if* $p \in F$ *and* $\alpha \in P$ *then* $\langle\alpha\rangle\, p \in F$,
4. $P_0 \subseteq P$,
5. *if* $\alpha \in P$ *and* $\beta \in P$ *then* $(\alpha \cup \beta) \in P$, $(\alpha;\beta) \in P$ *and* $\alpha^* \in P$,
6. *if* $p \in F$ *then* $p? \in P$.

Definition 6. *A structure is a triple* $D = (W, \tau, \delta)$ *where* W *is a set of* states, τ *assigns subsets of* W *to atomic formulas, and* δ *assigns subsets of* $W \times W$ *to atomic programs. The mappings* τ *and* δ *are extended inductively to determine the meaning of compound formulas and programs as follows:*

$\tau(true) = W$,
$\tau(false) = \emptyset$,
$\tau(\neg p) = \overline{\tau(p)}$,
$\tau(p \vee q) = \tau(p) \cup \tau(q)$,
$\tau(\langle\alpha\rangle\, p) = \{s \in W : \exists t\, ((s,t) \in \delta(\alpha) \wedge t \in \tau(p))\}$,
$\delta(\alpha;\beta) = \{(s,t) : \exists u\, ((s,u) \in \delta(\alpha) \wedge (u,t) \in \delta(\beta))\}$,
$\delta(\alpha \cup \beta) = \delta(\alpha) \cup \delta(\beta)$,
$\delta(p?) = \{(s,s) : s \in \tau(p)\}$,
$\delta(\alpha^*) = \delta(\alpha)^* =$
$\quad \{(s,t) : \exists k, s_0, s_1, \ldots, s_k (s_0 = s \wedge s_k = t \wedge (\forall 1 \leq i \leq k)(s_{i-1}, s_i) \in \delta(\alpha))\}$.

The presence of the Kleene star operator in the language of dynamic logic requires to expand fork algebras in order to obtain the interpretability result.

Definition 7. *A* closure AFA *(*CAFA *for short), is a structure* $\langle\, A,\ ^* \rangle$ *such that* A *is an* AFA, *and* * *satisfies the equations*

1. $R^* = 1\text{'} + R;R^*$,
2. $R^*;S;1 \leq S;1 + R^*;(\overline{S;1} \cdot R;S;1)$.

In order to show the validity of the second equation, let us analyze its meaning. The second-order formula

$$\forall R \forall S \forall x \exists y ((x R^* y \wedge y \in S) \rightarrow (x \in S \vee \exists z (x R^* z \wedge z \notin S \wedge \exists w (z R w \wedge w \in S))))$$

expresses that if a finite path exists in the graph induced by R, which connects x with an element $y \in S$, then, either x is already in S, or from x we can reach an object outside S which is next to an object in S. This is a desirable property of the operation *. Keeping in mind that right-ideal relations represent sets, it is easy to see that equation 2 in Def. 7 is equivalent to this second-order formula.

We define the mappings T_{DL} and T_P, mapping formulas and programs, as follows.

Definition 8. *In order to define mappings T_{DL} and T_P from dynamic logic formulas and compound programs, respectively, onto closure fork algebra with urelements terms, we first define recursively the mappings T'_{DL} and T_P by:*

$T'_{DL}(p_i) = P_i, (p_i$ *an atomic formula.*)
$T'_{DL}(true) = {}_{\mathsf{U}}1,$
$T'_{DL}(false) = 0,$
$T'_{DL}(\neg p) = 1'_{\mathsf{U}} ; \overline{T'_{DL}(p)},$
$T'_{DL}(p \vee q) = T'_{DL}(p) + T'_{DL}(q),$
$T'_{DL}(\langle R \rangle\, p) = T_P(R) ; T'_{DL}(p),$

$T_P(R_i) = R_i, (R_i$ *an atomic program.*)
$T_P(R \cup S) = T_P(R) + T_P(S),$
$T_P(R ; S) = T_P(R) ; T_P(S),$
$T_P(R^*) = T_P(R)^*,$
$T_P(p?) = T'_{DL}(p) \cdot 1'_{\mathsf{U}}.$
Next, we define the mapping T_{DL} by $T_{DL}(\alpha) = T'_{DL}(\alpha) + \overline{{}_{\mathsf{U}}1}$.

Using the mappings T_{DL} and T_P, in [30] the following theorem is proved.

Theorem 5. *Given a dynamic formula φ, we have*

$$\models_{DL} \varphi \qquad \Longleftrightarrow \qquad \vdash_{\mathsf{CAFA}} T_{DL}(\varphi)\ .$$

Interpretability of First-Order Dynamic Logic Dynamic logic has become a very useful tool in Computer Science, with direct applications in system specification. Here we show how to interpret first-order dynamic logic in an extension of the relational calculus of fork algebras. This allows us to: (a) incorporate the features of dynamic logic in a relational framework, and, (b) provide an equational calculus for reasoning in first-order dynamic logic.

Along this section we will assume a fixed (but arbitrary) finite signature $\Sigma = \langle\, s, A, F, P\,\rangle$, where s is a sort, $A = \{\, a_1, \ldots, a_n\,\}$ are the primitive action symbols, $F = \{\, f_1, \ldots, f_k\,\}$ are the function symbols, and $P = \{\, p_1, \ldots, p_m\,\}$ are the atomic predicate symbols. We will work in the monosorted case for simplicity, but the whole development extends straightforwardly to the manysorted case.

Definition 9. *The sets of* programs *and* formulas *on Σ are the smallest sets $Prg(\Sigma)$ and $For(\Sigma)$ satisfying:*

1. *$a \in Prg(\Sigma)$ for all $a \in A$.*
2. *If $r, s \in Prg(\Sigma)$, then $\{\, r^*, r \cup s, r ; s\,\} \subseteq Prg(\Sigma)$.*
3. *If $\alpha \in For(\Sigma)$, then $\alpha? \in Prg(\Sigma)$, and is called a* test *program.*

4. *The set of classical first-order atomic formulas on the signature Σ is contained in $For(\Sigma)$.*
5. *If $\alpha, \beta \in For(\Sigma)$ and x is a variable, then $\{\neg\alpha, \alpha \vee \beta, (\exists x)\,\alpha\} \subseteq For(\Sigma)$.*
6. *If $\alpha \in For(\Sigma)$ and $p \in Prg(\Sigma)$, then $\langle p \rangle\, \alpha \in For(\Sigma)$.*

As is standard in dynamic logic, states are valuations of the (state) variables. The set of states will be denoted by S. In the remaining part of this section we will assume a fixed (but arbitrary) structure $\mathcal{S} = \langle\, \mathbf{s}, m_{\mathcal{S}} \,\rangle$, where $\mathbf{s}$ is the *carrier* of the structure and $m_{\mathcal{S}}$ is the meaning function. The meaning function operates on functions and predicates as is standard in classical first-order logic. Given an action symbol $a \in A$, $m_{\mathcal{S}}(a) \subseteq S \times S$. Given a term t denoting an object from $\mathbf{s}$ and a state ν, $m_\nu(t)$ denotes the value of t in the state ν. When $\mathcal{S}$ is fixed, we will use just m instead of $m_{\mathcal{S}}$. The semantics of complex actions and formulas is given in the next definition. The notation $\mathcal{S}, \nu \models_{DL} \alpha$, is to be read "*the formula α is satisfied in the structure $\mathcal{S}$ by the state ν*". When $\mathcal{S}$ is clear from the context, we will use the notation $\nu \models_{DL} \alpha$ with the same meaning.

Definition 10. *The semantics of programs and formulas is given as follows.*

1. *If $a \in A$, then $m(a)$ is already defined.*
2. *If $a = b^*$, with $b \in Prg(\Sigma)$, then $m(a)$ is the reflexive-transitive closure of the binary relation $m(b)$.*
3. *If $a = b \cup c$, with $b, c \in Prg(\Sigma)$, then $m(a)$ is the union of the binary relations $m(b)$ and $m(c)$.*
4. *If $a = b;c$, with $b, c \in Prg(\Sigma)$, then $m(a)$ is the composition of the binary relations $m(b)$ and $m(c)$.*
5. *If $a = \alpha?$ with $\alpha \in For(\Sigma)$, then $m(a) = \{\, \langle \nu, \nu \rangle : \nu \models_{DL} \alpha \,\}$.*
6. *If $\varphi = p(t_1, \ldots, t_n)$ with $p \in P$, $\nu \models_{DL} \varphi$ if $\langle m_\nu(t_1), \ldots, m_\nu(t_n) \rangle \in m(p)$.*
7. *If $\varphi = \neg\alpha$, then $\nu \models_{DL} \varphi$ if $\nu \not\models_{DL} \alpha$.*
8. *If $\varphi = \alpha \vee \beta$, then $\nu \models_{DL} \varphi$ if $\nu \models_{DL} \alpha$ or $\nu \models_{DL} \beta$.*
9. *If $\varphi = (\exists x)\alpha$, then $\nu \models_{DL} \varphi$ if there exists $a \in \mathbf{s}$ such that $\nu[a/x] \models_{DL} \alpha$.*
10. *If $\varphi = \langle p \rangle\, \alpha$, then $\nu \models_{DL} \varphi$ if there exists a state ν' such that $\langle \nu, \nu' \rangle \in m(p)$ and $\nu' \models_{DL} \alpha$.*

In order to prove interpretability of first-order dynamic logic, we will need a more expressive class of algebras. We will define the class of omega closure fork algebras by extending CAFA with a new operator called *choice*, and a new equational proof rule. Because the theory of first-order dynamic logic is not recursively enumerable, CAFA with its recursively enumerable theory cannot be the target of the interpretation. In order to overcome this restriction, in Def. 11 we will define the calculus ω-CCFA by extending the the calculus for CAFA with an appropriate infinitary equational inference rule. In order to give a better understanding of the semantics of the calculus, in Def. 13 we define the class of *proper closure fork algebras*. In Thm. 6 we present a representation theorem showing that every model of ω-CCFA is isomorphic to some proper closure fork algebra.

Definition 11. *We define the calculus ω-CCFA by adding the following axioms and inference rules to the calculus of CAFA.*

$$x^{\diamond} ; 1 ; \breve{x^{\diamond}} \leq 1' , \qquad \textit{(Ax. 4)}$$

$$\breve{x^{\diamond}} ; 1 ; x^{\diamond} \leq 1' , \qquad \textit{(Ax. 5)}$$

$$1 ; \left(x \cdot x^{\diamond} \right) ; 1 = 1 ; x ; 1 . \qquad \textit{(Ax. 6)}$$

The inference rules for the calculus ω-CCFA are those of equational logic (see for instance [14, p. 94]) plus the following inference rule:

$$\frac{\vdash 1' \leq y \qquad x^i \leq y \vdash x^{i+1} \leq y}{\vdash x^* \leq y} \qquad (i \in \mathbb{N})$$

Definition 12. *A model of the identities provable in ω-CCFA will be called an* omega closure fork algebra*. The class of omega closure fork algebras is denoted by ω-CFA.*

Once the abstract algebras are defined, we will define the standard models and present the representation theorem, whose proof is given in [27].

Definition 13. *A* Proper Closure Fork Algebra *is a structure $\langle \mathfrak{A}, {}^{\diamond}, {}^{*} \rangle$ where $\mathfrak{A}$ is a proper fork algebra, * is reflexive-transitive closure, and ${}^{\diamond}$ is defined by the formula*

$$x^{\diamond} \subseteq x \text{ and } |x^{\diamond}| = 1 \quad \Longleftrightarrow \quad x \neq \emptyset \ .$$

Notice that $x^{\diamond}$ denotes an arbitrary pair in x. That is why $x^{\diamond}$ is called a choice operator*.*

Theorem 6. *Given $\mathfrak{A} \in \omega$-CFA, there exists $\mathfrak{B} \in$ PCFA such that $\mathfrak{A}$ is isomorphic to $\mathfrak{B}$.*

Given a subsequence, τ, of σ, of length k, by $\Pi_{\sigma,\tau}$ we denote the term

$$\delta_{\sigma}(v_{\tau(1)}) \nabla \cdots \nabla \delta_{\sigma}(v_{\tau(k)}) \ .$$

This term, given an object storing values for the variables whose indices occur in σ, builds an object storing values for the variables occurring in τ. Let τ and σ be disjoint and with $Length(\tau) = l_1$ and $Length(\sigma) = l_2$. Let $a = a_1 \star \cdots \star a_{l_1}$ and $b = b_1 \star \cdots \star b_{l_2}$ encode the values for the variables whose indices occur in τ and σ, respectively. $Merge_{\tau,\sigma}$, from the input $a \star b$, builds the encoding for the values of those variables whose indices occur either in τ or σ.

Definition 14. *The functions M_σ and T_σ are mutually defined by*

1. $M_\sigma(a) = (\mathit{\Pi}_{\sigma,\sigma_a} ; a \ \nabla \ \mathit{\Pi}_{\sigma,\sigma-\sigma_a}) \, ; Merge_{\sigma_a,\sigma-\sigma_a}$, *for each* $a \in A$,
2. $M_\sigma(R^*) = M_\sigma(R)^*$,
3. $M_\sigma(R \cup S) = M_\sigma(R) + M_\sigma(S)$,
4. $M_\sigma(R;S) = M_\sigma(R);M_\sigma(S)$,
5. $M_\sigma(\alpha?) = T_\sigma(\alpha) \cdot 1'$,
6. $T_\sigma(p(t_1,\ldots,t_n)) = (\delta_\sigma(t_1) \nabla \cdots \nabla \delta_\sigma(t_n)) \, ; p$,
7. $T_\sigma(\neg\alpha) = \overline{T_\sigma(\alpha)}$,
8. $T_\sigma(\alpha \vee \beta) = T_\sigma(\alpha) + T_\sigma(\beta)$,
9. $T_\sigma((\exists v_i)\, \alpha) = \mathit{\Delta}_{\sigma,i} ; T_{\sigma\oplus i}(\alpha)$,
10. $T_\sigma(\langle p \rangle\, \alpha) = M_\sigma(p) ; T_\sigma(\alpha)$.

As usual, a formula α will be called a *sentence* if it does not contain any free occurrences of variables. The next theorem states the interpretability of presentations of theories from *DL* as equational theories in ω-CCFA. A detailed proof is given in [27].

Theorem 7. *Let $\mathit{\Gamma} \cup \{\, \varphi \,\}$ be a set of sentences. Then,*

$$\mathit{\Gamma} \models_{DL} \varphi \qquad \Longleftrightarrow \qquad \left\{\, T_{\langle\rangle}(\gamma) = 1 : \gamma \in \mathit{\Gamma} \,\right\} \vdash_{\omega\text{-}\mathsf{CCFA}} T_{\langle\rangle}(\varphi) = 1 \ .$$

3.6 Fork Algebras in Program Derivation

As a consequence of Thm. 1, the first-order theories of AFA and PFA are the same, and thus a natural semantics can be attributed to first-order formulas over abstract relations in terms of binary relations. This is a very important property in a calculus for program construction. The equivalence between the first-order theories of PFA and AFA guarantees that any first-order property valid for proper fork algebras can be proved syntactically from the axioms describing abstract fork algebras. This has a direct application in program construction. Let us consider an intermediate step in a derivation of an algorithm from a relational specification S_0. The derivation has a shape

$$S_0, S_1, \ldots, S_k,$$

where for all i, $1 \leq i \leq k$, S_i is obtained from $S_0, \ldots, S_{i-1}$ by means of the derivation rules. If S_k is still not the algorithm we are looking for, then further steps must be performed. If resorting to thinking about binary relations shows that a valid first-order property allows S_k to evolve to a new expression E (which is closer to the intended algorithm), then the representation theorem guarantees that a syntactic proof $S_k, S_{k+1}, \ldots, E$ exists, allowing us to reach the formula E from S_k. This shows that the heuristics arising from considering *concrete* binary relations can be employed throughout the process of program derivation using the rules and axioms of the calculus for *abstract* fork algebras. Another important property stems from the fact that only a finite number of axioms are necessary for describing the class of abstract fork algebras. Thus, the syntactic proofs mentioned above can be more easily performed with the assistance of a computer system.

Regarding the expressiveness of fork algebras, it was proved in Section 3.4 that first-order theories can be interpreted as equational theories in fork algebras. Theorem 3 shows that a wide class of problems (at least those that can be described in first-order logic) can be specified in the equational calculus of fork algebras. Moreover, the abstract relational specification can be obtained algorithmically from the first-order specification by using the mapping $\mathcal{I}$.

A Methodology for Program Construction In this section we present the outlines of a methodology for program construction based on fork algebras using generic algorithms (program schemes).

The starting point of the methodology is a formal specification of the problem to be solved. In this case, we will use first-order logic with equality because it is a simple formal language that is taught in most computer science courses.

Once a first-order specification of a problem is given, a relational specification must be obtained. In order to obtain this specification, we can proceed in one of the following two ways. Applying Thm. 3, from a first-order specification φ and using the mapping $\mathcal{I}$ we will obtain a relational term $\mathcal{I}(\varphi)$ that captures the meaning of problem P. Unfortunately, the term resulting from applying the mapping $\mathcal{I}$ is not always very adequate with respect to the process of program derivation. The second method consists of reducing the first-order formula φ into an equation e_φ using the set-theoretical definition of the relational operators.

Once a relational specification is obtained, we will choose a design strategy that will guide the process of deriving an algorithm from this specification. In programming in general, examples of design strategies include case analysis, trivialization, divide-and-conquer, backtracking, and many more [1, 10, 52, 55, 56]. In the methodology presented here, the first-order language of fork algebras will be used to express such design strategies (recall that from Thm. 1 and the discussion following it, formulas from the first-order theory of fork algebras have a standard semantics in terms of concrete binary relations). 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:

$$C\_A(R, R_1, \ldots, R_k) \iff \bigwedge_{1 \leq i < j \leq k} \mathit{Dom}\,(R_i) \cdot \mathit{Dom}\,(R_j) = 0 \wedge R = \sum_{i=1}^{k} R_i \,. \quad (14)$$

Formula (14) is to be read as follows:

'Problem R is solved by case-analysis using problems $R_1, \ldots, R_k$'.

Notice that (14) provides the means to solve problem R, that is, given an input a for R there is to find R_i such that $a \in \mathit{Dom}\,(R_i)$ and then compute $R_i(a)$.

Other strategies, such as trivialization (a particular instance of case analysis where one of the subproblems is assumed to be easy to solve), divide-and-conquer, or backtracking have been formalized in this way.

Each strategy comes with an associated explanation about how to construct a program solving the original problem.

Notice that strategies are in general formulas of the form

$$\mathit{Strat}(R, X_1, \ldots, X_n) \qquad \iff \qquad \mathit{Strat\_Definition}(R, X_1, \ldots, X_n), \qquad (15)$$

where *Strat* is a $(n+1)$-ary predicate symbol, and *Strat_Definition* is a formula on the relational variables R, $X_1, \ldots, X_n$, involving previously defined strategies. Deriving an algorithm A for solving a problem P whose relational specification is $S(P_1, \ldots, P_k)$ using a strategy defined as in (15), consists of finding relational terms $T_1(P_1, \ldots, P_k), \ldots, T_n(P_1, \ldots, P_k)$ such that

$$\begin{aligned} &Theory(D_1), \ldots, Theory(D_m), S(P_1, \ldots, P_k) \vdash_{\mathsf{AFA}} \\ &\qquad Strat\_Definition(P, T_1(P_1, \ldots, P_k), \ldots, T_n(P_1, \ldots, P_k)). \quad (16) \end{aligned}$$

In (16), $Theory(D_1), \ldots, Theory(D_m)$ are relational specifications of the domains of the problem [7–9], and the symbol $\vdash_{\mathsf{AFA}}$ denotes first-order logic entailment under the theory of AFA.

The algorithms characterized by the strategies are as a matter of fact fork algebraic equations. Thus, in order to find terms $T_1(P_1, \ldots, P_k), \ldots, T_n(P_1, \ldots, P_k)$ we will resort to equational reasonings using the axioms of fork algebras, plus those equations describing the domains $D_1, \ldots, D_m$. The general strategy we will use for deriving recursive algorithms will be *Unfolding/Folding* [16].

The terms $T_1(P_1, \ldots, P_k), \ldots, T_n(P_1, \ldots, P_k)$ required in (16), and found as described in the previous paragraph, are either algorithms if they are built with algorithmic combinators, or can be considered as relational specifications of simpler problems.

4 Current Research

4.1 The **A**$r_{\mathbf{g}}$*entum* Project

The results presented in Sections 3.4 and 3.5 constitute the foundations of the **A**$r_{\mathbf{g}}$*entum* Project. **A**$r_{\mathbf{g}}$*entum* is a CASE tool with relational foundations under development by the group of Relational Methods, at the department of computer science of the University of Buenos Aires. Rather than using a single monolithic language for software specification, it uses different logics for modeling different views of systems. Thus, a system specification becomes a collection of theories coming from different logics. Using the interpretability results for these logics, the theories are translated to a uniform (regarding the language) relational specification. Once a relational specification is obtained, different tools such as model checkers or theorem provers can be applied in order to verify the relational specification. For a graphical description of **A**$r_{\mathbf{g}}$*entum*, see Fig. 5. UML diagrams located at the top of Fig. 5 are mapped to specifications in different logics. Notice that more than one arrow can leave from a single diagram. This is because one diagram can contain information of different nature. For instance, a class-diagram contains structural information that can be specified in classical first-order logic. It can also contain behavioral information as pre and post conditions of methods, that can be captured using dynamic logic. The theory presentations holding this information are located at the spheres. Notice that several arrows can target the same sphere. This is because information of a similar nature can appear scattered through the diagrams. The arrows originating at the spheres map logical specifications to a relational specification (located in the box targeted by the arrows). The homogeneous specification can later be analyzed using tools (the lower boxes) which can be plugged into **A**$r_{\mathbf{g}}$*entum*.

The mappings from the spheres to the fork calculus have been implemented for classical first-order logic and first-order dynamic logic. Also have been implemented a theorem prover for first-order dynamic logic as an extension of the theorem prover PVS [51], and a model checker for relational specifications.

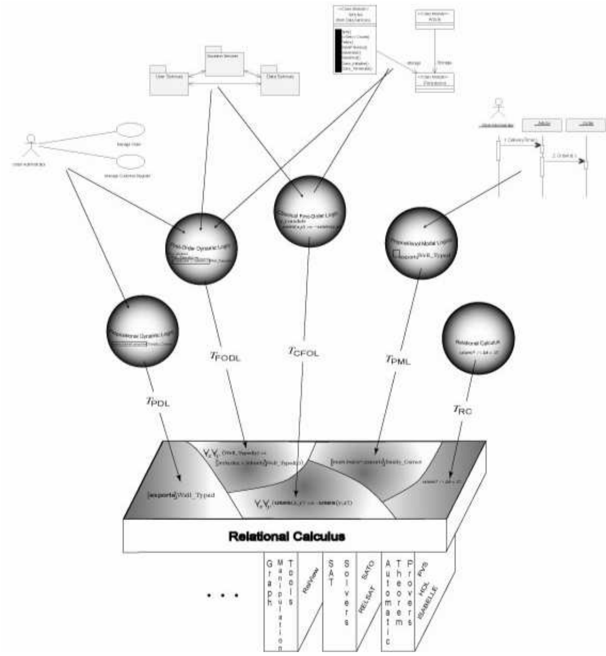

Fig. 5. Architecture of **Ar**g*entum*

4.2 Interpretability of First-Order Temporal Logics

This section presents validity-preserving mappings translating first-order formulas from the temporal logic *LTL* to the language of omega closure fork algebras. An essential property of the translation is that states in traces are modeled in the same way the state is modeled in the translation of classical first-order logic and first-order dynamic logic.

In the remaining part of this section we will present the syntax and semantics of the first-order linear time temporal logic *LTL*, as well as the mapping and the interpretability theorem.

The logic *FOLTL* is defined over sets of variables $\{v_k\}_{k\in\mathcal{K}}$. Terms are built as in classical first-order logic. The set of well-formed terms is denoted by *TerFOLTL*.

Definition 15. *We define the set ForFOLTL of the FOLTL well-formed formulas on the first-order signature Σ as the smallest set F satisfying:*

- *If p_i ($i \in \mathcal{I}$) is an n-ary atomic predicate and $t_1, \ldots, t_n \in$ TerFOLTL, then $p(t_1, \ldots, t_n) \in F$,*
- *If $\alpha, \beta \in F$ and v_k ($k \in \mathcal{K}$) is a variable, then*

$$\{\, \neg\alpha, \alpha \vee \beta, \oplus\alpha, \alpha \mathsf{U} \beta, (\exists v_k)\alpha, (\forall v_k)\alpha \,\} \subseteq F \ .$$

Definition 16. *Given a signature $\Sigma = \langle\, s, \{f_j\}_{j\in\mathcal{J}}, \{p_i\}_{i\in\mathcal{I}}\,\rangle$, a Σ-structure is a structure $\mathcal{A} = \left\langle\, s^{\mathcal{A}}, \{f_j^{\mathcal{A}}\}_{j\in\mathcal{J}}, \{p_i^{\mathcal{A}}\}_{i\in\mathcal{I}}\,\right\rangle$ such that:*

- *$s^{\mathcal{A}}$ is a nonempty set.*
- *if f_j ($j \in \mathcal{J}$) is an n-ary function symbol, then $f_j^{\mathcal{A}} : (s^{\mathcal{A}})^n \to s^{\mathcal{A}}$.*
- *if p_i ($i \in \mathcal{I}$) is an n-ary predicate symbol, then $p_i^{\mathcal{A}} \subseteq (s^{\mathcal{A}})^n$.*

Given a signature Σ, we will assume a fixed (but arbitrary) Σ-structure $\mathcal{A}$. The semantics of *FOLTL* formulas is defined over a Kripke structure K of the form $\langle\, \mathcal{A}, St, St_0, T\,\rangle$, where St is the set of *states* (valuations of the variables on $s^{\mathcal{A}}$), $St_0 \subseteq St$ is the set of *initial states*, and $T \subseteq St \times St$ is the *transition relation*. The transition relation T is assumed to be *complete*; that is, every state has at least one successor.

Given a Kripke structure K, the set of paths of K is denoted by Δ_K. A path $s \in \Delta_K$ is an infinite sequence $s_0, s_1, \ldots$ such that $s_i \in St$ and $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. We denote by s^i the suffix of s starting at position i. Similarly, we denote by s_i the i-th state in the path s. A v_i–variant of a state s ($i \in \mathcal{K}$) is a state $\hat{s}$ that agrees with s in the value of the state variables v_j ($j \in \mathcal{K}, j \neq i$). This concept generalizes to traces as follows. A trace $\hat{\pi} = \hat{s_0}, \hat{s_1}, \ldots, \hat{s_n}, \ldots$ is a v_i–variant of a trace $\pi = s_0, s_1, \ldots, s_n, \ldots$ if $\hat{s_j}$ is a v_i–variant of s_j for all $j \geq 0$.

The semantics of terms agrees with the semantics of terms in classical first-order logic. In the next definition we present the satisfiability relation for *FOLTL* formulas.

Definition 17. *Given a Kripke structure $K = \langle\, \mathcal{A}, St, St_0, T\,\rangle$, formulas $\alpha, \beta \in$ ForFOLTL, and $s \in \Delta_K$, the semantics of a FOLTL formula is defined recursively as follows:*

- $K, s \models_{FOLTL} p_i(t_1, \ldots, t_n)$ *iff* $(V(t_1)(s_0), \ldots, V(t_n)(s_0)) \in p_i^{\mathcal{A}}$,
- $K, s \models_{FOLTL} \neg\alpha$ *iff* $K, s \not\models \alpha$,
- $K, s \models_{FOLTL} \alpha \vee \beta$ *iff* $K, s \models \alpha$ *or* $K, s \models \beta$,
- $K, s \models_{FOLTL} \oplus\alpha$ *iff* $K, s^1 \models \alpha$,
- $K, s \models_{FOLTL} \alpha \mathsf{U} \beta$ *iff there exists* $i \geq 0$ *such that* $K, s^i \models \beta$, *and for all* j ($0 \leq j < i$), $K, s^j \models \alpha$,
- $K, s \models_{FOLTL} (\exists v_j)\alpha$ *iff* $K, \hat{s} \models \alpha$, *for some* $\hat{s}$, *a* v_j*–variant of* s.

A formula is satisfied in a Kripke structure K if it is satisfied along a path $s_0, s_1, \ldots \in \Delta_K$ such that $s_0 \in St_0$. A formula is valid in a Kripke structure K if it is satisfied along all paths $s_0, s_1, \ldots \in \Delta_K$ such that $s_0 \in St_0$.

Defining the translation for a first-order temporal language with function symbols $\{f_j\}_{j\in\mathcal{J}}$ and atomic proposition symbols $\{p_i\}_{i\in\mathcal{I}}$, requires extending the language of closure fork algebras with new constants $\mathbf{St}$, $\mathbf{T}$, $\mathbf{St_0}$, $\mathbf{tr}$, and families of constants $\{\mathbf{F}_j\}_{j\in\mathcal{J}}$, $\{\mathbf{P}_i\}_{i\in\mathcal{I}}$ and $\{\mathbf{V}_k\}_{k\in\mathcal{K}}$.

There are two usual ways to represent sets as binary relations: using partial identities (i.e., relations contained in the identity relation), or using right-ideal relations. Right-ideal relations relate each element in their domain to every element in the universe. Thus, the range provides no information. A right-ideal relation can be used to model the set provided by its domain.

In the following paragraphs we will present axioms characterizing the meaning of the added constants. The partial identity $\mathbf{St}$ will model the set St. Similarly, relation $\mathbf{St_0}$ is a partial identity modeling the set St_0. Relation $\mathbf{T}$ models the accessibility relation T. Relation $\mathbf{tr}$ models the set of traces. The constants $\mathbf{F}_j$ ($j \in \mathcal{J}$) model the meaning of the function symbols. Similarly, relations $\mathbf{P}_i$ ($i \in \mathcal{I}$) will model the meaning of predicate symbols.

$$\mathbf{St} = 1'_{\mathsf{U}} \otimes \cdots \otimes 1'_{\mathsf{U}} \quad (|\mathcal{K}| \text{ times}), \tag{17}$$

$$\mathbf{St_0} \leq \mathbf{St}, \tag{18}$$

$$\mathit{Dom}\,(\mathbf{T}) = \mathbf{St}, \tag{19}$$

Formula (17) establishes that the states are built as k-tuples of urelements. Formula (18) establishes that $\mathbf{S_0}$ is a subset of the set of states. Formula (19) establishes that $\mathbf{T}$ is a total (and therefore complete) relation on the set of states.

For each function symbol f, with arity n, we add the equations:

$$\breve{\mathbf{F}} ; \mathbf{F} \leq 1'_{\mathsf{U}}, \tag{20}$$

$$(\underbrace{1'_{\mathsf{U}} \otimes \cdots \otimes 1'_{\mathsf{U}}}_{n \text{ times}}) ; \mathbf{F} = \mathbf{F}. \tag{21}$$

Equation (20) establishes that $\mathbf{F}$ is a functional relation, and (21) establishes that $\mathbf{F}$ expects a n tuple as input and produces an urelement as output.

For each predicate symbol P, with arity n, we add the equation:

$$(\underbrace{1'_{\mathsf{U}} \otimes \cdots \otimes 1'_{\mathsf{U}}}_{n \text{ times}}) ; \mathbf{P} ; 1 = \mathbf{P}. \tag{22}$$

Formula (22) establishes that $\mathbf{P}$ a right-ideal relation, therefore representing a set. $\mathbf{P}$ represents the set of n tuples that satisfy predicate P.

Since the semantics of temporal formulas is defined in terms of traces, we will model the notion of trace in a fork algebra. Given a fork algebra $\mathfrak{A}$, we model traces in $\mathfrak{A}$ with elements from $U_{\mathfrak{A}}$ as the ones described by Fig. 6. The next definition provides a relational characterization of traces.

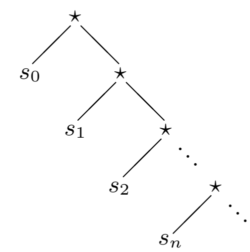

Fig. 6. Infinite right degenerate trees pattern.

The relation **tr**, characterizing the traces (paths) in a closure fork algebra is defined by the following equations:

$$\mathbf{tr} \leq 1', \tag{23}$$

$$\breve{\pi}\,;\mathbf{tr}\,;\pi = \mathbf{St}, \tag{24}$$

$$\mathbf{tr} \leq \mathbf{St} \otimes \mathbf{tr}, \tag{25}$$

$$\mathbf{tr}\,;\rho = Ran\left(\pi \nabla \left(\mathbf{T} \otimes \rho\right)\right); \rho\,;\mathbf{tr}. \tag{26}$$

Formula (23) states that **tr** is a partial identity (a set). Formula (24) (together with (17)) establishes that states in a trace are k-tuples of urelements. Finally, Formulas (25) and (26) establish that traces are infinite, T-related, sequences.

The relations $\mathbf{V}_i$ allow us to build the v_i–variants of a trace. They are defined as follows:

$$\mathbf{V}_i = \nu_X \begin{pmatrix} Rep_{n,i} & \\ \otimes & ;\mathbf{tr} \\ X & \end{pmatrix},$$

where ν is the largest fixed point operator, and $Rep_{n,i}$ denotes the binary relation that, when provided with a state $a_1 \star \cdots \star a_i \star \cdots \star a_n$, returns all the states obtained by substituting the value of a_i. For instance, for $n = 3$,

$$Rep_{3,1} = {}_{\mathsf{U}}1_{\mathsf{U}} \otimes \left(1' \otimes 1'\right),$$
$$Rep_{3,2} = 1' \otimes \left({}_{\mathsf{U}}1_{\mathsf{U}} \otimes 1'\right),$$
$$Rep_{3,3} = 1' \otimes \left(1' \otimes {}_{\mathsf{U}}1_{\mathsf{U}}\right).$$

A simple proof by induction on n (that essentially uses the monotonicity of $T(X)$) shows that $\mathbf{V}_i$ is a lower bound of the chain $1, T(1), \ldots, T^n(1), \ldots$. If we add the rule

$$\frac{y \leq T_i^j(1) \vdash y \leq T_i^{j+1}(1)}{\vdash y \leq \mathbf{V}_i} \quad (VarRule_i)$$

then $\mathbf{V}_i$ is indeed the largest lower bound (the infimum) of the chain.

Definition 18. *We define the calculus* ω-CCFA' *as the extension of* ω-CCFA *obtained by adding equations (17)–(26) as axioms, and the rules* $VarRule_i$ $(1 \leq i \leq k)$.

Terms are translated as in classical first-order logic. In Def. 19 we present a translation of *FOLTL* formulas to fork terms.

Definition 19. *We define the translation* $T_{FOLTL} : ForFOLTL(\Sigma) \to AFATerms$, *mapping formulas from FOLTL to terms in fork algebras, as follows:*

$$\begin{aligned}
T_{FOLTL}(p_i(t_1, \ldots, t_n)) &= \pi ; \left(\delta_{FOLTL}(t_1) \nabla \cdots \nabla \delta_{FOLTL}(t_n)\right) ; \mathbf{P}_i \\
T_{FOLTL}(\neg\alpha) &= \mathbf{tr} ; \overline{T_{FOLTL}(\alpha)} \\
T_{FOLTL}(\alpha \vee \beta) &= T_{FOLTL}(\alpha) + T_{FOLTL}(\beta) \\
T_{FOLTL}(\oplus\alpha) &= \rho ; T_{FOLTL}(\alpha) \\
T_{FOLTL}(\alpha \mathsf{U} \beta) &= \left(Dom\left(T_{FOLTL}(\alpha)\right) ; \rho\right)^{*} ; T_{FOLTL}(\beta) \\
T_{FOLTL}\left((\exists v_i)\, \alpha\right) &= \mathbf{V}_i ; T_{FOLTL}(\alpha)
\end{aligned}$$

The next theorem presents the interpretability result for the logic *LTL*. It shows that it is possible to replace semantic reasoning in *LTL* by equational reasoning in ω-CCFA'.

Theorem 8. *Let* $\alpha \in ForFOLTL(\Sigma)$. *Then,*

$$\models_{FOLTL} \alpha \quad \iff \quad \vdash_{\omega\text{-}\mathsf{CCFA}'} Dom\left(\pi ; \mathbf{St_0}\right) ; \mathbf{tr} ; T_{FOLTL}(\alpha) = Dom\left(\pi ; \mathbf{St_0}\right) ; \mathbf{tr} ; 1 \,.$$

4.3 Fork and Arrow Logics

We will now outline some ideas of a current research connecting fork algebras and modal logics. As mentioned in Section 1, modal logics are convenient formalisms for several purposes, modalities being simpler than quantifiers, but generally have limited expressive power [12]. Arrow logic, for instance, is a modal logic intended to deal with arrows (e.g., transitions) and can handle sequential composition, but not parallelism [44].

In the sequel, we will indicate how the addition of fork to arrow logic provides an extension where one can deal with parallelism and synchronization [5, 6, 20]. We will use a notation close to that of the present paper, rather than the more traditional one in modal circles.

Arrow logic amounts to a modal version of relation algebras [44, 12]. We will first outline the basic ideas of arrow logic (*AL*, for short).

The formulas of *AL* are built from a set P of propositional symbols, as follows: $\alpha ::= p \mid 1\text{'} \mid \overline{\alpha'} \mid \alpha' \cdot \alpha'' \mid \breve{\alpha}' \mid \alpha' ; \alpha''$.[16] Other symbols can be defined, such as the duals $\bar{\breve{}}$ and $\underline{;}$[17].

[16] A more traditional notation uses $\iota\delta$ and $\otimes$ for 1' and $\breve{}$.

[17] We define $\alpha^{\bar{\breve{}}} := \overline{\breve{\overline{\alpha}}}$ and $\beta \,\underline{;}\, \gamma := \overline{\overline{\beta} ; \overline{\gamma}}$.

The semantics of *AL* is given by frames and models. A *square arrow frame* is given by a square set S (of states): $S = U \times U$.[18] A *square arrow model* $\mathfrak{M}$ consists of a square arrow frame and a valuation $V : P \to 2^S$ (assigning to each propositional letter in P a set of of states).

Now, *satisfaction* at a state $s = \langle a, b\rangle \in S$ is defined inductively as usual. For instance, some clauses are

[p] $(\mathfrak{M}, \langle a, b\rangle) \models p$ iff $\langle a, b\rangle \in V(p)$,
[1'] $(\mathfrak{M}, \langle a, b\rangle) \models 1'$ iff $a = b$,
[$\breve{}$] $(\mathfrak{M}, \langle a, b\rangle) \models \breve{\alpha}$ iff $(\mathfrak{M}, \langle b, a\rangle) \models \alpha$,
[;] $(\mathfrak{M}, \langle a, b\rangle) \models \beta;\gamma$ iff, for some $c \in U$, $(\mathfrak{M}, \langle a, c\rangle) \models \beta$ and $(\mathfrak{M}, \langle c, b\rangle) \models \gamma$.

If we use $\mathfrak{M}[\alpha]$ for the set $\{ s \in S : (\mathfrak{M}, s) \models \alpha \}$ of states defined by formula α, we see that we have an algebra of relations.[19] We have a clear correspondence between relation algebra equations and *AL* formulas.

Fork arrow logic (*FAL*, for short) extends *AL* by fork ∇. The formulas of *FAL* are built as before, adding the clause $\alpha ::= \alpha' \nabla \alpha''$. We introduce the projections $pi := (1' \nabla 1)^{\smile}$ and $rho := (1 \nabla 1')^{\smile}$, as well as the dual $\underline{\nabla}$ by $\beta \underline{\nabla} \gamma := \overline{\overline{\beta} \nabla \overline{\gamma}}$.

The semantics of *FAL* is given by the appropriate frames and models. A *square fork arrow frame* $\mathfrak{F}$ consists of a square arrow frame and an injective function $\star : U \times U \to U$. A *square fork arrow model* $\mathfrak{M}$ consists of a square fork arrow frame $\mathfrak{F}$ and a valuation $V : P \to 2^S$.

Now, *satisfaction* at a state $s = \langle a, b\rangle \in S$ is defined by extending the previous definition by the following clause:

[∇] $(\mathfrak{M}, \langle a, b\rangle) \models \beta \nabla \gamma$ iff, for some $c, d \in U$, $(\mathfrak{M}, \langle a, c\rangle) \models \beta$, $(\mathfrak{M}, \langle c, d\rangle) \models \gamma$ and $c \star d = b$.

We can axiomatize *FAL* by a normal modal system [12], consisting of

- the *FAL* versions of the fork arrow logic equations,
- distribution axioms for the dual modalities (e.g., $\alpha \underline{\nabla} (\beta \to \gamma) \to (\alpha \underline{\nabla} \beta) \to (\alpha \underline{\nabla} \gamma))$,
- necessitation rules for the dual modalities (e.g., from β infer $\beta \underline{\nabla} \gamma$)).

This axiom system is sound and complete [5] and it axiomatizes squares in *FAL* [20].

We can internalize the above *FAL* semantics in first-order logic by considering a set $\underline{P}$ of binary predicates corresponding to the propositional letters in P, as well as a binary operation $\star$. We then have similar *FAL* models $\mathfrak{M}$ and first-order

[18] A general arrow frame is a structure $\langle S, ;, 1', \breve{} \rangle$.
[19] We have, e.g., $\mathfrak{M}[1'] = Id$, $\mathfrak{M}[\breve{\alpha}] = (\mathfrak{M}[\alpha])^{\smile}$ and $\mathfrak{M}[\beta;\gamma] = \mathfrak{M}[\beta] \circ \mathfrak{M}[\gamma]$.

structures $\mathfrak{T}$ (with $\underline{p}^{\mathfrak{T}} = V(p)$) corresponding to each other. We define the *square fork arrow translation* Q_u^z, for a pair of variables u and z, simulating the above inductive definition of satisfaction. For instance, some clauses are as follows.

[p] $Q_u^z(p) := \underline{p}(u, z)$,
[1'] $Q_u^z(1') := u = z$,
[;] $Q_u^z(\beta;\gamma) := \exists v(Q_u^v(\beta) \wedge Q_v^z(\gamma))$, where v is a new variable,
[∇] $Q_u^z(\beta\nabla\gamma) := \exists v \exists w(Q_u^v(\beta) \wedge Q_u^w(\gamma) \wedge v \star w = z)$, where v and w are new variables.

We can also define a reverse translation R_u^z, from first-order formulas to *FAL* formulas, much as before (cf. Section 3.4.). We can then see the expressiveness of *FAL* formulas [20].

1. For similar *FAL* model $\mathfrak{M}$ and first-order structure $\mathfrak{T}$,
   (a) for every *FAL* formula α: $(\mathfrak{M}, \langle a, b\rangle) \models \alpha$ iff $\mathfrak{T} \models Q_u^z(\alpha)\ [u/a, z/b]$, i.e. we have equal defined relations $\mathfrak{M}[\alpha] = \mathfrak{T}[Q_u^z(\alpha)]$,
   (b) for every first-order formula φ: $\mathfrak{T} \models \varphi\ [u/a, z/b]$ iff $(\mathfrak{M}, \langle a, b\rangle) \models R_u^z(\varphi)$, i.e. we have equal defined relations $\mathfrak{T}[\varphi] = \mathfrak{M}[R_u^z(\varphi)]$.
2. For every *FAL* formula α: $\vdash_{FAL}\ \alpha \leftrightarrow R_u^z(Q_u^z(\alpha))$.
3. For every first-order formula φ: $Inj(\star) \vdash\ \varphi \leftrightarrow Q_u^z(R_u^z(\varphi))$.

5 Conclusions and Further Research Directions

In this paper we have presented an overview of research done in the past within the field of fork algebras. We also devote special attention to the description of current research, as well as mention in the remaining of this section some future research lines. Hopefully, this paper will serve as a reference for researchers interested in fork algebras, and for those already working on fork algebras and looking for new research lines within the field.

Automatic Analysis of Relational Specifications We have started some foundational research and tool development to this end. A promising research direction is bounded model-checking of relational specifications. The bounds are necessary because the theory of fork algebras is undecidable. There are two approaches that we are studying. One consists on verifying specifications using proper fork algebras, and the other uses abstract fork algebras.

Verification using proper fork algebras requires bounds on the cardinality of the set of urelements. Notice that urelements are used in modeling user defined objects, while splitting elements are used in order to cope with the lack of variables over individuals in the language of fork algebras. Verifying that an assertion follows from a specification involving relational constants $C_1, \ldots, C_k$, amounts to finding binary relations $R_1, \ldots, R_k$ such that the assignment $C_1 := R_1, \ldots, C_k := R_k$ satisfies the specification, but refutes the assertion. Generating all possible k-tuples of relations is usually unfeasible, and therefore the verification problem consists on generating as few valuations as possible, while making sure that counterexamples are not incorrectly overlooked. Different strategies can be implemented in order to reduce the number of valuations that have to be generated, and this is an important research area.

Verification using abstract fork algebras attempts to solve the same problem. The difference is that $R_1, \ldots, R_k$ rather than being binary relations, are elements from an abstract fork algebra. The added complexity, in this case, is that interesting fork algebras are infinite. This can be solved by adding a new bound on the depth of pairing. The advantage is that the new method is strictly more expressive than checking with concrete relations. This follows from the fact that there are finite and representable relation algebras that are representable on infinite sets. Therefore, for some problems about infinite relations, as for instance dense linear orders, properties can be verified in small, finite, abstract fork algebras.

Formalization of Component Based Software Development Interfaces, services, components and connectors are concepts that have a clear relational content. Services can be seen as the building blocks for component based software. We can model services as relations for which we provide relational specifications. Services can be appropriately conjoined in order to obtain a relational formalization of components. Components are related using connectors. In a relational setting, connectors can be modeled with terms built with projections, allowing us to connect services from two or more components.

Fork Algebras, Allegories and Situation Theory Another research line concerns connections of fork algebras with other theories, e.g., Allegory and Situation Theory. In both cases, one can notice connections with relation algebras, which suggests investigating the introduction of fork in such contexts. Allegories [22] appear as generalizations of categories, motivated by relaxing the functional morphisms of the latter to relational arrows. Situation Theory [3] was introduced to handle information, in general, and information flow in Channel Theory [2]. Here, the flow of information through a channel suggests input-output pairs, also the operations and axioms on channels remind one of algebras of relations. It is quite natural to consider fork to model parallel flows and synchronization.

6 Acknowledgements

Marcelo Frias wishes to thank Ms. Claudia Rojas for several conversations that deeply influenced this paper.

References

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *Data Structures and Algorithms* (1983), (Addison-Wesley, Reading, MA).
2. Barwise, J., *Constraints, channels and the flow of information*, Aczel, P., Israel, D., Katagiri, Y., and Peters, S. (Eds.), Situation Theory and its Applications, volume 3 CSLI, Stanford, 1993, pp. 3–27.
3. Barwise, J. and Perry, J., *Situation and Attitudes*, MIT Press, Cambridge, Ma, 1983.
4. Baum, G. A., Frias, M. F., Haeberer, A. M. and Martínez López, P. E., *From Specifications to Programs: A Fork–algebraic Approach to Bridge the Gap*, in Proceedings of MFCS'96, LNCS 1113, (Springer-Verlag, 1996, pp. 180–191).
5. Benevides, M. R. F. and Veloso, P. A. S., *Axiomatization and completeness for fork modal logic*, XII Encontro Brasileiro de L-gica'99 Itatiaia, May 1999, pp. 87–94.
6. Benevides, M. R. F., Freitas, R. P., Veloso, P. A. S., Veloso, S. R. M. and Viana, J. M., *Axiomatization and completeness for fork arrow logic*, IV Workshop de Mtodos Formais (WMF'2001) Rio de Janeiro, Oct. 2001, pp. 5.1–5.12.
7. Berghammer, R., *Relational Specifications of Data Types and Programs*, Technical Report 9109, Fakultät für Informatik, Universität der Bundeswehr München, 1991.
8. Berghammer, R. and Schmidt, G., *Relational Specifications*, in C. Rauszer (Ed.), Algebraic Logic, Banach Center Publications vol. 28, Polish Academy of Sciences, 1993, pp. 167–190.
9. Berghammer, R., Gritzner T.F., and Schmidt, G., *Prototyping Relational Specifications Using Higher-Order Objects*, in Heering, J., Meinke, K., Möller, B. and Nipkow, T. (Eds.), Higher-Order Algebra, Logic and Term Rewriting, 1st. International Workshop, HOA'93, LNCS 816, (Springer-Verlag, 1993, pp. 56–75).
10. Bird, R. and de Moor O., *Algebra of Programming* (1997), (Prentice Hall).
11. Booch, G., Jacobson, I. and Rumbaugh, J., *The Unified Modeling Language User Guide*, The Addison-Wesley Object Technology Series, 1998.
12. Blackburn, P., de Rijke, M. and Venema, Y., *Modal Logic* (2001), (Cambridge University Press, Cambridge).
13. Brink, C., Kahl, W. and Schmidt, G. (Eds.), *Relational Methods in Computer Science* (1997), (Springer Wien–New York).
14. Burris, S., and Sankappanavar H. P., *A Course in Universal Algebra*, Graduate Texts in Mathematics, Springer Verlag, 1981.
15. Buszkowski, W. and Orlowska, E., *Indiscernibility-Based Formalisation of Dependencies in Information Systems*. In Orlowska, E. (Ed.), Incomplete Information: Rough set analysis, (Physica Verlag, 1997).
16. Darlington, J., *Applications of Program Transformation to Program Synthesis*, in Proceedings of the International Symposium on Proving and Improving Programs, Arc-et-Senans, France, July 1-3, 1975, pp. 133–144.
17. Demri, S., Orlowska, E. and Rewitzky, I., *Towards reasoning about Hoare relations*, Annals of Mathematics and Artificial Intelligence vol. 12 (1994), pp. 265–289.
18. Demri, S. and Orlowska, E., *Logical Analysis of Demonic Nondeterministic Programs*, Theoretical Computer Science vol. 166 (1–2) (1996).
19. Elustondo, P. M., Veloso, P. A. S., Haeberer, A. M. and Vázquez, L. A., *Program development in the algebraic theory of problems*, 18as. Jornadas Argentinas de Informática e Investigación Operativa, Buenos Aires, Aug. 1989, pp. 2.2–2.32.

20. Freitas, R. P., Veloso, P. A. S., Viana, J. M., Veloso, S. R. M., and Benevides, M. R. F., *Expressive power of fork arrow logic*, Martini, A., and D'eharbe, D. (Eds.), Proc. 5th Workshop on Formal Methods (WMF'2002) Porto Alegre, Oct. 2002, pp. 89–99.
21. Freitas, R. P., Viana, J. M., Benevides, M. R. F., Veloso, S. R. M., and Veloso, P. A. S., *Squares in fork modal logic*, Journal of Philosophical Logic vol. 32, No. 2 (2003), pp. 343–355.
22. Freyd, P. and Scedrov, A., *Categories, Allegories*, North-Holland, Amsterdam, 1991.
23. Frias, M. F., *Fork Algebras in Algebra, Logic and Computer Science*, World Scientific, Advances in Logic, Vol. 2, 2002.
24. Frias, M. F., *Independence of the Axiomatization of Fork*, Algebra Universalis, vol. 39, pp. 211–215, 1998.
25. Frias, M. F., Baum, G. A. and Haeberer, A. M., *Adding Design Strategies to Fork Algebras*, in Proceedings of Perspectives of System Informatics, LNCS 1181, (Springer-Verlag, 1996), pp. 214–226.
26. Frias M.F., Baum G.A. and Haeberer A.M., *Representability and Program Construction within Fork Algebras*, Logic Journal of the IGPL, Vol. 6, No. 2, (Oxford University Press, 1998), pp. 229–259.
27. Frias M. F., Baum G. A. and Maibaum T. S. E., *Interpretability of First- Order Dynamic Logic in a Relational Calculus*, in Proceedings of RelMiCS 6, LNCS 2561, Springer-Verlag, 2002.
28. Frias, M. F., Baum, G. A., Haeberer, A. M. and Veloso, P. A. S., *Fork Algebras are Representable*, Bulletin of the Section of Logic vol. 24, No. 2 (1995), University of Lódź, pp. 64–75.
29. Frias, M. F., Haeberer, A. M. and Veloso, P. A. S., *A Finite Axiomatization for Fork Algebras*, Logic Journal of the IGPL vol. 5, No. 3, (Oxford University Press, 1997) pp. 311–319.
30. Frias M.F. and Orlowska E., *Equational Reasoning in Non Classical Logics*, Journal of Applied Non Classical Logics, Vol. 8, No. 1-2 (1998), pp. 27–66.
31. 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), pp. 281–298.
32. Haeberer, A. M and Veloso, P. A. S., *Why software development is inherently non-monotonic: a formal justification*, Trappl, R. (Ed.) Cybernetics and Systems Research, Scientific Publ. Corp, London, 1990, pp. 51–58.
33. Haeberer, A. M and Veloso, P. A. S., *Partial relations for program derivation: adequacy, inevitability and expressiveness*, Möller, B. (Ed.) Constructing Programs from Specifications, North-Holland, Amsterdam, 1991, pp. 319–371.
34. Harel, D., *Dynamic Logic*, Handbook of Philosophical Logic, Vol. II, (D. Reidel Publishing Company, 1984), pp. 497–604.
35. Henkin, L., Monk, D. and Tarski, A., *Cylindric Algebras, Part I* (1971), Studies in Logic and the Foundations of Mathematics, vol. 64, (North-Holland).
36. Henkin, L., Monk, D. and Tarski, A., *Cylindric Algebras, Part II* (1985), Studies in Logic and the Foundations of Mathematics, vol. 115, (North-Holland).
37. Herment, M. and Orlowska, E., *Handling information logics in a graphical proof editor*, Computational Intelligence vol. 11, No. 2, (1995), pp. 297–322.
38. Jackson D., *Alloy: A Lightweight Object Modelling Notation*, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 11, Issue 2 (April 2002), pp. 256-290.
39. Jónsson, B., and Tarski, A., *Boolean Algebras with Operators, PART II*, American Journal of Mathematics vol. 74 (1952), pp. 127–162.
40. Kripke, S., *Semantical analysis of modal logic I*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik vol. 9, (1963), pp. 67–96.
41. Kripke, S., *Semantical analysis of intuitionistic logic*. In: Crossley, J. N. and Dummett, M. A. (Eds.) Formal Systems and Recursive Functions, North Holland, 1965.
42. Maddux, R. D., *Introductory Course on Relation Algebras, Finite-Dimensional Cylindric Algebras and their Interconnections*, Colloquia Mathematica Societatis János Bolyai vol. 54, Algebraic Logic, Budapest, Hungary, 1988, North-Holland, pp. 361–392.

43. Manna Z. and Pnueli A., *The Temporal Logic of Reactive and Concurrent Systems – Specification*, Springer-Verlag, 1991.
44. Marx, M., Polos, L. and Venema, Y. (Eds.), *Arrow Logic and Multi-Modal Logic*, CSLI Public., Stanford, 1996.
45. Mikulás, S., Sain, I. and Simon, A., *Complexity of the Equational Theory of Relational Algebras with Projection Elements.* Bulletin of the Section of Logic vol. 21, No. 3 (1992), University of Lódź, pp. 103–111.
46. Monk, J. D., *On Representable Relation Algebras*, Michigan Mathematical Journal, vol. 11, 1964, pp. 207–210.
47. Orlowska, E., *Relational interpretation of modal logics.* In: Andreka, H., Monk, D. and Nemeti, I. (Eds.), Algebraic Logic. Colloquia Mathematica Societatis Janos Bolyai vol. 54, (North-Holland, 1988), pp. 443–471.
48. Orlowska, E., *Relational proof system for relevant logics*, Journal of Symbolic Logic vol. 57 (1992), pp. 1425–1440.
49. Orlowska, E., *Relational semantics for nonclassical logics: Formulas are relations.* In: Wolenski, J. (Ed.), Philosophical Logic in Poland, (Kluwer, 1994), pp. 167–186.
50. Orlowska, E., *Relational proof systems for modal logics.* In: Wansing, H. (Ed.), Proof Theory of Modal Logics, (Kluwer, 1996), pp. 55–77.
51. Owre S., Shankar N., Rushby J.M. and Stringer-Calvert D.W.J., *PVS System Guide*, SRI International, version 2.4 edition, December 2001.
52. Partsch, H. A., *Specification and Transformation of Programs. A Formal Approach to Software Development* (1990), Texts and Monographs in Computer Science, (Springer-Verlag).
53. Polya, G., *How to Solve it: a new aspect of the mathematical method*, Princeton University Press, Princeton, 1945 (2. edn. 1956, repr. 1971).
54. Sain, I. and Németi, I., *Fork Algebras in Usual as well as in Non-well-founded Set Theories*, Studia Logica Library (a special volume dedicated to the memory of H.Rasiowa), to appear; extended abstract appeared in Bulletin of the Section of Logic, University of Lódź, Vol. 24, N. 3-4, 1995, pp. 158–168, pp. 182–192.
55. Smith, D. R., *Top-Down Synthesis of Divide-and-Conquer Algorithms*, Artificial Intelligence vol. 27 (1985), pp. 43–96.
56. Smith, D. R., *Applications of a Strategy for Designing Divide-and-Conquer Algorithms*, Science of Computer Programming vol. 8 (1987), Elsevier Science Publishers B. V., pp. 213–229.
57. Tarski, A., *On the Calculus of Relations*, Journal of Symbolic Logic vol. 6 (1941), pp. 73–89.
58. Tarski, A. and Givant, S., *A Formalization of Set Theory without Variables* (1987), American Mathematical Society Colloquium Publications, vol. 41, American Mathematical Society.
59. Veloso, P. A. S., *Outlines of a mathematical theory of general problems*, Philosophia Naturalis vol. 2/4, No. 1 (1984), University of Lódź, pp. 354–362.
60. Veloso, P. A. S., *Is fork set-theoretical?*, Bulletin of the Section of Logic vol. 26, No. 1 (1997), University of Lódź, pp. 20–30.
61. Veloso, P. A. S., *Characterisations for fork algebras and their relational reducts*, Bulletin of the Section of Logic vol. 26, No. 3 (1997), University of Lódź, pp. 144–155.
62. Veloso, P. A. S., *On the independence of the axioms for fork algebras*, Bulletin of the Section of Logic vol. 26, No. 4 (1997), University of Lódź, pp. 197–209.
63. Veloso, P. A. S., *On eight independent equational axiomatisations for fork algebras*, Bulletin of the Section of Logic vol. 27, No. 3 (1998), University of Lódź, pp. 117–129.
64. Veloso, P. A. S. and Haeberer, A. M., *Software development: a problem-theoretic analysis and model*, Shriver, B. D. (Ed.) Proc. 22nd Hawaii International Conference on System Science, vol. II: Software Track, Scientific Publ. Corp, Kona, HI , Jan. 1989, pp. 200–209.
65. Veloso, P. A. S. and Haeberer, A. M., *A Finitary Relational Algebra for Classical First-Order Logic*, Bulletin of the Section of Logic vol. 20, No. 2 (1991), University of Lódź, pp. 52–62.

Journal on Relational Methods in Computer Science, Vol. 1, 2004, pp. 181 - 216
Received by the editors April 16, 2004, and, in revised form, December 02, 2004.

Published on December 10, 2004.