On the Expressive Power of Fork Arrow Logic

R.P. de Freitas[1], P.A.S. Veloso[1,2], J.P. Viana[1,3],
S.R.M. Veloso[1,2], and M.R.F. Benevides[1,2]

[1] Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ
[2] Instituto de Matemática, Universidade Federal do Rio de Janeiro
[3] Instituto de Matemática, Universidade Federal Fluminense

Abstract. We compare fork arrow logic, an extension of arrow logic, and its natural first-order counterpart (the correspondence language). Arrow logic is a modal logic for reasoning about arrow structures, its expressive power is limited to a bounded fragment of first-order logic. Fork arrow logic is obtained by adding to arrow logic the fork modality (related to parallelism and synchronization). As a result, fork arrow logic has the same expressive power of its first-order correspondence language, so both can express the same input/output behavior of processes.

1 Introduction

We show that fork arrow logic and its natural counterpart in first-order logic have the same expressive power. Thus both systems can express the same input/output behaviors.

Fork arrow logic is an extension of arrow logic. The latter is a modal logic used to formalize reasoning about arrow structures, i.e., structures having arrows as individuals and whose basic concepts are distinguished arrows and relations on arrows. Given the close relationship between arrow structures and relation algebras, arrow logic is strongly connected to the equational theory of relation algebras [Jon82]. The expressive power of arrow logic is limited to a bounded fragment of first-order logic [Ven91].

An arrow formula under square semantics defines a binary relation as a set of ordered pairs. Such a relation may also be regarded as the input/output behavior of a process. Interpreting processes as relations, we may have non-determinism. The apparatus of arrow logic provides means for combining relations/processes. Some of these have direct programming interpretation, while others are useful in specifying their behaviors. In this manner, we may specify the behavior of a process by combining behaviors of component processes. Some features, however, seem absent, namely those related to parallelism, synchronization, etc. Together with the above limitation this interpretation suggests that arrow logic is not an adequate formalism for specification and development of programs, in a more realistic context.

Fork arrow logic is a modal logic related to the equational theory of fork algebras. A fork algebra is a relation algebra with an extra operator called *fork* and the proper version extends an algebra of binary relations by the fork induced

by an underlying pairing function. The extension of the relational calculus by fork arose in computing as a formalism for specification and derivation of (non-deterministic) programs (with parallelism).

The development of fork arrow logic can be seen in a more general context, shown in the Brink-Venema diagram of Fig. 1 of [FVVVB]. In this paper we show that fork mades arrow logic more expressive, in the sense that every input/output behavior that one can express in first-order logic can also be expressed in fork arrow logic. We believe that this result makes arrow logic more realistic, mainly in connection with program development and analysis. Examples illustrating the application of fork algebras that can be adapted to fork arrow logic can be found in [FBH97,HFBV97,Vel99] and in the bibliography therein.

2 Arrow Logic

In this section we briefly review arrow logic (AL) under square semantics.

Here we consider AL as a modal system designed to talk and reason about relational structures whose individuals are ordered pairs and the basic relations are the relations of composition and reversion of ordered pairs. Introduced in [vBen91,vBen94] and developed in [Ven91,Mar95], AL is a widely applicable system, being able to formalize notions from mathematics, computer science, linguistics, cognitive science, semantics, etc. An overview of AL with an extensive bibliography can be found in [MPM96,MV97].

The basic ideas underlying AL are as follows: each propositional letter is interpreted as a set of arrows and each modality is interpreted as an accessibility relation, which explains how arrows can be decomposed.

2.1 Syntax and Square Semantics of Arrow Logic

The *alphabet* of AL consists of a set P of *propositional letters*, whose elements are denoted by $p, q, \ldots$, indexed or not; the *Boolean operators* $\neg$ and $\wedge$; and the *Peircean operators* $\iota\delta, \otimes$, and $\circ$. The *formulas* of AL with set P of propositional letters (AL[P]) are defined by:

$$\alpha := p \mid \iota\delta \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2 \mid \otimes\alpha \mid \alpha_1 \circ \alpha_2.$$

We consider the following operators as defined: $\top$, $\rightarrow$, $\leftrightarrow$.

Semantically, arrows are considered as concrete objects, constructed from more basic ones: their *end-points*. The basic relations on arrows are defined by explicit reference to these end-points. An *arrow square frame* (over nonempty set $\mathcal{U}$) is a 4-tuple $\mathcal{F} = \langle S, C, R, I\rangle$, such that:
i) $S = \mathcal{U}^2$;
ii) $C \subseteq S \times S \times S$ where $C(a_1, b_1)(a_2, b_2)(a_3, b_3)$ iff $a_1 = a_2$, $b_1 = b_3$ and $b_2 = a_3$;
iii) $R \subseteq S \times S$ where $R(a_1, b_1)(a_2, b_2)$ iff $a_1 = b_2$ and $b_1 = a_2$;
iv) $I \subseteq S$ where $I(a, b)$ iff $a = b$.

An *arrow square model* is a pair $\mathcal{M} = \langle \mathcal{F}, V\rangle$, where $\mathcal{F}$ is an arrow square frame and $V : \mathsf{P} \rightarrow 2^S$ is a valuation function, i.e., $V(p)$ is a binary relation over

$\mathcal{U}$ for each $p \in \mathsf{P}$. A *rooted arrow square model* is a pair $\langle \mathcal{M}, (a,b) \rangle$, where $\mathcal{M}$ is an arrow square model and (a,b) is a distinguished arrow.

Satisfaction of arrow formulas in a rooted arrow square model is defined by:
i) $\mathcal{M}, (a,b) \models p$ iff $(a,b) \in V(p)$,
ii) $\mathcal{M}, (a,b) \models \iota\delta$ iff $a = b$,
iii) $\mathcal{M}, (a,b) \models \neg\alpha$ iff $\mathcal{M}, (a,b) \not\models \alpha$,
iv) $\mathcal{M}, (a,b) \models \alpha \wedge \beta$ iff $\mathcal{M}, (a,b) \models \alpha$ and $\mathcal{M}, (a,b) \models \beta$.
v) $\mathcal{M}, (a,b) \models \otimes\alpha$ iff $\mathcal{M}, (b,a) \models \alpha$,
vi) $\mathcal{M}, (a,b) \models \alpha \circ \beta$ iff $\mathcal{M}, (a,c) \models \alpha$ and $\mathcal{M}, (c,b) \models \beta$, for some $c \in \mathcal{U}$.

The *behavior* of an arrow formula α in an arrow square model $\mathcal{M}$, denoted by $\mathcal{M}[\alpha]$, is the binary relation defined by $\mathcal{M}[\alpha] := \{(a,b) : \mathcal{M}, (a,b) \models \alpha\}$.

Lemma 1. *If $\mathcal{M}$ is an arrow square model over $\mathcal{U}$, then*[1]*:*

(i) $\mathcal{M}[p] = V(p)$ *(ii)* $\mathcal{M}[\iota\delta] = Id_{\mathcal{U}}$
(iii) $\mathcal{M}[\neg\alpha] = \overline{\mathcal{M}[\alpha]}$ *(iv)* $\mathcal{M}[\alpha \wedge \beta] = \mathcal{M}[\alpha] \cap \mathcal{M}[\beta]$
(v) $\mathcal{M}[\otimes\alpha] = \mathcal{M}[\alpha]^{-1}$ *(vi)* $\mathcal{M}[\alpha \circ \beta] = \mathcal{M}[\alpha] \mid \mathcal{M}[\beta]$

2.2 Square Translation

An axiomatization for AL with the square semantics is given in [Ven91], where the expressive power of AL and its natural counterpart in first-order logic (FOL) are compared. By examining relational algebraic ideas in the modal context Venema proved that, under the square semantics, AL has the same expressive power as the limited fragment of FOL whose formulas have exactly three variables, two being free. For our proposes it suffices to recall part of these ideas.

Given a set P of propositional letters whose elements are denoted by $p, q, r, \ldots$ indexed or not, we denote by $\underline{\mathsf{P}}$ the set whose elements are denoted by $P, Q, R, \ldots$. indexed or not. The *first-order correspondence arrow language* associated to $\underline{\mathsf{P}}$ (FOL[$\underline{\mathsf{P}}$]) is the first-order language whose non-logical symbols are the elements of $\underline{\mathsf{P}}$ as binary predicate symbols.

Each arrow model for AL[P] corresponds to a first-order structure for FOL[$\underline{\mathsf{P}}$]. In fact, if $\mathcal{M} = \langle S, C, R, I, V \rangle$ is a model for AL[P], with $S = \mathcal{U}^2$, the associated first-order structure is $\mathcal{A} = \langle \mathcal{U}, P^{\mathcal{A}}, Q^{\mathcal{A}}, \ldots \rangle$, where $P^{\mathcal{A}} = V(p), Q^{\mathcal{A}} = V(q), \ldots$. In this case we say that $\mathcal{M}$ and $\mathcal{A}$ are *similar*.

Given variables x, y, the *arrow square translation* associated to x, y is the function ST_{xy} from formulas of AL[P] to formulas of FOL[$\underline{\mathsf{P}}$], defined by:
i) $ST_{xy}(p) := P(x,y)$,
ii) $ST_{xy}(\iota\delta) := x \approx y$,
iii) $ST_{xy}(\neg\alpha) := \neg ST_{xy}(\alpha)$,
iv) $ST_{xy}(\alpha \wedge \beta) := ST_{xy}(\alpha) \wedge ST_{xy}(\beta)$,
v) $ST_{xy}(\otimes\alpha) := ST_{yx}(\alpha)$, and
vi) $ST_{xy}(\alpha \circ \beta) := \exists z(ST_{xz}(\alpha) \wedge ST_{zy}(\beta))$, where z is a new variable.

[1] Given a set $\mathcal{U}$ and R, S relations over $\mathcal{U}$, we define $Id_{\mathcal{U}} = \{(a,b) \in \mathcal{U}^2 : a = b\}$, $\overline{R} = \{(a,b) \in \mathcal{U}^2 : (a,b) \notin R\}$, $R \cap S = \{(a,b) \in \mathcal{U}^2 : (a,b) \in R$ and $(a,b) \in S\}$, $R^{-1} = \{(a,b) \in \mathcal{U}^2 : (b,a) \in R\}$, and $R|S = \{(a,b) \in \mathcal{U}^2 :$ there is $c \in \mathcal{U}$ such that $(a,c) \in R$ and $(c,b) \in S\}$.

Theorem 1 (ST and similar models). *For each formula α of* AL[P]*, rooted square arrow model $\langle \mathcal{M}, (a,b)\rangle$ and first-order model $\mathcal{A}$ similar to $\mathcal{M}$, we have*[2]*:*
$\mathcal{M}, (a,b) \models \alpha$ *iff* $\mathcal{A} \models ST_{xy}(\alpha)[x \mapsto a, y \mapsto b]$.

3 Parallelism and Coding

The behavior of an AL formula under square semantics is a set of ordered pairs. Such behavior may be regarded as the input/output behavior of a process. Given a process, its behavior is the binary relation connecting inputs (initial states) to outputs (final states). The apparatus of AL provides means for combining such behaviors. Some of these (like sequential composition) have direct programming interpretation, while others (like negation) are useful in specifying behaviors of processes. In this manner, we may specify the behavior of a process by combining behaviors of component processes. Some features, however, seem absent, namely those related to parallelism, synchronization, etc.

As a simple example of parallelism, consider integers represented by sign and modulus (absolute value). To double such an integer, one keeps its sign and doubles its modulus, to square it, one assigns positive to sign and squares its modulus. In either case, we are applying two processes in parallel: one to the sign and another one to the modulus. Here, each integer is represented by a record, consisting of sign and modulus; as such, parallel processes may be regarded as manipulating ordered pairs.

As a matter of fact, one does not need actual cartesian-like records; some coding for them is enough. For instance, for natural numbers, one might consider a Gödel-like coding $\star : \mathbb{N}^2 \to \mathbb{N}$, given by, say, $m \star n := 2^m \cdot (2n+1)$, coding pair (m,n) of naturals by the single natural $m \star n \in \mathbb{N}$. The intuition is that, as long as one can recover the given arguments from the coded pair, one does not care about the particular coding schema adopted. It can be thought of as an internal matter left to the system. Thus, by a *pair-coding* on universe $\mathcal{U}$, we shall mean an injective function $\star : \mathcal{U}^2 \to \mathcal{U}$, coding each pair (a,b) of elements of $\mathcal{U}$ by the single element $a \star b \in \mathcal{U}$.

Now, we have new means of combining process behaviors. *Parallel product* // corresponds to parallel execution, as such $P//Q := \{(a \star b, c \star d) \in \mathcal{U}^2 : (a,c) \in P \text{ and } (b,d) \in Q\}$.

$$\begin{pmatrix} a \\ \star \\ b \end{pmatrix} \xrightarrow{P//Q} \begin{pmatrix} c \\ \star \\ d \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} a \xrightarrow{P} c \\ \text{and} \\ b \xrightarrow{Q} d \end{pmatrix}$$

Another natural means of combining processes comes from the idea of feeding a common input to two processes. This new operation, called *fork*, produces

[2] Notice that this comparison involves classifying a and b (or the corresponding variables x and y) as input and output. This classification is apparently arbitrary, but here this is not crucial, because we can invert the classification by resorting to the transposal (which is the behavior of the modality $\otimes$).

relation $P \angle_\star Q := \{(a, b \star c) \in \mathcal{U}^2 : (a, b) \in P \text{ and } (a, c) \in Q\}$.

$$a \longrightarrow \begin{cases} a & \xrightarrow{P} \begin{pmatrix} b \\ \star \\ c \end{pmatrix} \\ a & \xrightarrow{Q} \end{cases}$$

We can also introduce some special processes. *Duplication* $2_\mathcal{U}$ produces two copies of the input. Then $2_\mathcal{U} := \{(a, b \star c) \in \mathcal{U}^2 : a = b = c\}$.

$$a \xrightarrow{2_\mathcal{U}} \begin{pmatrix} a \\ \star \\ a \end{pmatrix}$$

We also have processes for extracting components: the left and right *extractors* $\pi_\mathcal{U} := \{(a \star b, a) \in \mathcal{U}^2 : a, b \in \mathcal{U}\}$ and $\rho_\mathcal{U} := \{(a \star b, b) \in \mathcal{U}^2 : a, b \in \mathcal{U}\}$.

$$a \xleftarrow{\pi_\mathcal{U}} \begin{pmatrix} a \\ \star \\ b \end{pmatrix} \xrightarrow{\rho_\mathcal{U}} b$$

In the theory of operating systems the concept of fork is essential for understanding the difference between a program and a process, since fork operator generates children processes within the program. In addition, when one considers arrows as input/output of transition states, the fork relation provides synchronization and parallelism of transitions. The comments above suggest that the AL apparatus should be extended in order to express these programming ideas. First, the universe $\mathcal{U}$ should have coded pairs of its elements, i.e., it is to be closed under a coding function $\star$. So, we no longer have $\mathcal{U}$ as an unstructured set of elements. Instead, we will be dealing with a universe with structure, having objects such as $a, b, a \star b, a \star (a \star b), (a \star b) \star a$, etc. Second, we have some new, structural, operations on relations that appear to be convenient and natural on a structured universe. We shall take fork as primitive since, as we shall see in Sect. 4, the others are relationally definable from it.

4 Fork Arrow Logic

We extend the syntax and semantics of AL to obtain fork arrow logic (FAL).

4.1 Syntax and Square Semantics for Fork Arrow Logic

The *alphabet* of FAL with set P of propositional letters consists of the alphabet of AL[P] augmented by the symbol $\triangledown$. The *formulas* of FAL with set P of propositional letters (FAL[P]) are defined by the rules of AL[P] extended by the new rule $\alpha := \alpha_1 \triangledown \alpha_2$.

We freely use the standard abbreviations as in AL and consider the following modalities as defined: $2 := \iota\delta \triangledown \iota\delta$, $\pi := \otimes(\iota\delta \triangledown \top)$, $\rho := \otimes(\top \triangledown \iota\delta)$, and $\alpha \parallel \beta := (\pi \circ \alpha) \triangledown (\rho \circ \beta)$.

We need the notion of a frame having appropriate type for the fork modal language, and since we will be dealing with arrows with fork, we intend to interpret formulas into fork frames over a structured universe.

A *pair-coding structure* $\langle \mathcal{U}, \star \rangle$ is a nonempty set $\mathcal{U}$ with a pair-coding function $\star : \mathcal{U}^2 \to \mathcal{U}$. The *fork relation* induced by $\star$ is the ternary relation $F_\star$ over $\mathcal{U}^2$ such that $F_\star := \{((a,b),(c,d),(e,f)) : a = c = e \text{ and } b = d \star f\}$. A *fork square frame* over $\langle \mathcal{U}, \star \rangle$ is a frame $\mathcal{F} = \langle S, F_\star, C, R, I \rangle$ where:
i) $\langle \mathcal{U}, \star \rangle$ is a pair-coding structure,
ii) $\langle S, C, R, I \rangle$ is an arrow square frame over $\mathcal{U}$, and
ii) $F_\star$ is the fork relation induced by $\star$.

Note that, to specify a fork frame, one needs only a pair-coding structure $\langle \mathcal{U}, \star \rangle$, since the relations $F_\star, C, R, I$ are defined in a standard way.

The notions of fork square model and rooted fork square model are analogous to that of square arrow model and rooted square arrow model. Satisfaction of a fork formula by an arrow in a fork square model is defined as in AL, with the following extra clause:

$\mathcal{M}, (a,b) \models \alpha \bigtriangledown \beta$ iff $\exists b_1, b_2 \in \mathcal{U}$ such that $b = b_1 \star b_2$, $\mathcal{M}, (a, b_1) \models \alpha$ and $\mathcal{M}, (a, b_2) \models \beta$.

The behavior of a formula $\alpha \in \mathrm{AL}[\mathsf{P}]$ under a fork square model $\mathcal{M}$, denoted by $\mathcal{M}[\alpha]$, is defined as in AL, but observe that, in this case, the elements of $\mathcal{M}[\alpha]$ are ordered pair of elements such as $a, b, a \star b, (a \star b) \star a, a \star (b \star b)$, etc.

Besides clauses similar to that in Lemma 1, we also have:

Lemma 2. *If $\mathcal{M}$ is a fork square model over $\langle \mathcal{U}, \star \rangle$, then:*

(i) $\mathcal{M}[\alpha \bigtriangledown \beta] = \mathcal{M}[\alpha] \angle_\star \mathcal{M}[\beta]$
(ii) $\mathcal{M}[2] = 2_\mathcal{U}$
(iii) $\mathcal{M}[\pi] = \pi_\mathcal{U}$
(vi) $\mathcal{M}[\rho] = \rho_\mathcal{U}$
(v) $\mathcal{M}[\alpha \parallel \beta] = \mathcal{M}[\alpha] // \mathcal{M}[\beta]$

4.2 Square Translation

An axiomatization for FAL with the square semantics is given in [FVVVB]. In the sequel, we shall study the expressive power of FAL.

We shall define a first-order language where the fork formulas will be translated. Besides the predicate symbols to translate the propositional letters, the language also has a function symbol to translate the coding function[3].

Let P be a set of propositional letters. The $\star$ *first-order correspondence language* associated to P ($\mathrm{FOL}[\underline{\mathsf{P}}, \star]$) is the first-order language whose non-logical symbols are the elements of $\underline{\mathsf{P}}$ as binary predicate symbols and a binary function symbol $\star$.

As in AL, each fork square model corresponds to a first-order structure. If $\mathcal{M} = \langle S, F_\star, C, R, I, V \rangle$ is a square model for $\mathrm{FAL}[\mathsf{P}]$ over $\langle \mathcal{U}, \star \rangle$, the associated

[3] We use the same notation for the $\star$ symbol and for its realization.

structure is $\mathcal{A} = \langle \mathcal{U}, \star, P^{\mathcal{A}}, Q^{\mathcal{A}}, \ldots \rangle$, where $P^{\mathcal{A}} = V(p), Q^{\mathcal{A}} = V(q), \ldots$. In this case we say that $\mathcal{M}$ and $\mathcal{A}$ are *similar*.

Given two individual variables x, y from FOL[$\underline{\mathsf{P}}, \star$], the *fork square translation* associated to x, y is the function ST_{xy} that maps formulas of FAL[P] to formulas of FOL[$\underline{\mathsf{P}}, \star$], defined as in AL with the following extra rule:

$$ST_{xy}(\alpha \bigtriangledown \beta) := \exists zw(y \approx z \star w \wedge ST_{xz}(\alpha) \wedge ST_{xw}(\beta)), \text{ where } z, w \text{ are new.}$$

Theorem 2 (ST and similar models). *For each formula α of* FAL[P]*, rooted fork square model $\langle \mathcal{M}, (a, b) \rangle$, and first-order model $\mathcal{A}$ similar to $\mathcal{M}$, we have:*

$$\mathcal{M}, (a, b) \models \alpha \text{ iff } \mathcal{A} \models ST_{xy}(\alpha)[x \mapsto a, y \mapsto b].$$

5 Expressive Power of Fork Arrow Logic

In the preceding section we have seen that, much as for AL, the behavior of each fork formula can be described by a first-order formula with exactly two free variables. In this section we will establish the converse: each input/output behavior described by a first-order formula with exactly two free variables can also be described by a fork formula. We shall admit first-order formulas with arbitrary number of quantified variables, in contrast with the case of AL. For this purpose, we will provide a reverse translation RT_{xy} from formulas of FOL[$\underline{\mathsf{P}}, \star$] with exactly x and y as free variables to formulas of FAL[P], such that $\mathcal{A}[\varphi] = \mathcal{M}[RT_{xy}(\varphi)]$, whenever first-order model $\mathcal{A}$ and fork square model $\mathcal{M}$ are similar.

5.1 Main Ideas

The basic idea is that we have the Boolean connectives in both languages, first-order equality will be handled by the identity modality $\iota\delta$, and existential quantification (which may be regarded as a search) will be simulated by means of the projections.

Now, we wish to admit first-order formulas with arbitrary number of quantifiers. So, for inductive purposes, it will be convenient to consider first-order formulas with any (non-zero) number of free variables. Thus, we will first construct a general translation GT from formulas of FOL[$\underline{\mathsf{P}}, \star$] to formulas of FAL[P]. The preceding goal brings about a problem: the behavior of a fork formula is a binary relation, whereas that of a first-order formula with n free variables is an n-ary relation. Even if one classifies these variables as input and output, one of these categories will have more than one variable. So to compare fork formulas and first-order formulas, we will classify the free variables of the first-order formula as input and output and then compactify them by means of $\star$.

Also, to code a set of variables we will employ a tree-like term constructed from them by $\star$, e.g. the set $\{x, y, z\}$ can be coded by the term $(x \star y) \star z$. The idea

is that the term $(x \star y) \star z$ gives all the information given by the set $\{x, y, z\}$[4]. Finally, to define a unique term, we must order the variables.

5.2 Preliminary concepts

Consider a first-order language. We shall use the notations $\mathsf{vr}(t)$, and $\mathsf{vr}(\varphi)$, for the set of variables occurring in term t, respectively formula φ. Also $\mathsf{fv}(\varphi)$ stands for the set of variables with free occurrences in φ. We will use $\varphi(x/t)$ for the formula obtained by replacing each free occurrence of variable x by t in φ.

We shall order the variables of the first-order language in an arbitrary way. This ordering shall remain fixed throughout the constructions. Thus, when we write $x < y$ or $y > x$ we mean that x *precedes* y in this fixed ordering. Also, for a finite set $\underline{x}$ of variables, we have the *first* and *last* variables in $\underline{x}$ (denoted by $\mathsf{min}(\underline{x})$ and $\mathsf{Max}(\underline{x})$, respectively), and by $\mathsf{nxt}(\underline{x})$ we mean the first variable above all those in $\underline{x}$.

As an example of these ideas, we will need a special *ordered alphabetic variant* of existential formulas, namely $\exists y' \varphi(y')$, where y' is the next variable after all those free or bound in $\exists y \varphi(y)$. For this purpose, we let $\varphi^y := \varphi(y/\mathsf{nxt}(\mathsf{vr}(\varphi)))$.[5]

It is well-known that every first-order formula is equivalent to one in term-reduced form: with atomic subformulas of the forms $x \approx y$, $f(x_1, \ldots, x_n) \approx y$ and $R(x_1, \ldots, x_m)$ [EFT84]. In our case, with ordered variables, we have:

Lemma 3 (Ordered term-reduced form). *Every formula of* FOL$[\underline{\mathsf{P}}, \star]$ *is equivalent to one in term-reduced form with atomic subformulas of the forms $x \approx x$ (for any variable x), $x \approx y$ (for any two distinct variables x and y), $x \star y \approx z$ (for any three distinct variables $x < y < z$), $P(x, y)$ (for any two distinct variables $x < y$).*

Proof. The cases not contemplated by the ordering can be handled by equality with new variables selected as required. □

The code of a nonempty finite set $\underline{x} = \{x_1, \ldots, x_m\}$ of variables will be the $\star$ of the variables in order, i.e., $\mathsf{K}(\underline{x}) = (\cdots(x_1 \star x_2) \star \cdots) \star x_m$ if $x_1 < x_2 < \cdots < x_m$. Thus, we define the *variable code* by $\mathsf{K}(\{x\}) := x$ and $\mathsf{K}(\underline{x} \cup \{y\}) := \mathsf{K}(\underline{x}) \star y$ whenever $\mathsf{Max}(\underline{x}) < y$.

Now, consider $\underline{x} = \{x_1, \ldots, x_m\}$ with $x_1 < x_2 < \cdots < x_m$. The code $\mathsf{K}(\underline{x}) = (\cdots(x_1 \star x_2) \star \cdots) \star x_m$ is a tree-like term from which we can extract each portion by means of a fork formula.

[4] For instance assigning values a, b and c to x, y and z amounts to assigning the value $(a \star b) \star c$ to the term $(x \star y) \star z$, and, conversely, since each value assigned to the term $(x \star y) \star z$ must be of the form $(a \star b) \star c$, we can recover values to be assigned to x, y and z.

[5] For instance, consider formula $\exists y(x \star y \approx z)$ with $\mathsf{vr}(x \star y \approx z) = \{x, y, z\}$. If the ordering is $x < y < z < w$ (with no variable between z and w), then $\mathsf{nxt}(\mathsf{vr}(x \star y \approx z)) = w$; so $(x \star y \approx z)^y = x \star y \approx z(y/w) = x \star w \approx z$. Hence the ordered alphabetic variant is $\exists w(x \star w \approx z)$.

We can use compositions of projections to extract variables. To extract x_j from $\mathsf{K}(\underline{x})$, we can define the *variable-extracting* formula $\varepsilon(\underline{x} \hookrightarrow x_j)$ by: $\varepsilon(\underline{x} \hookrightarrow x_j) := \rho$ if $x_j = x_m$ and $\varepsilon(\underline{x} \hookrightarrow x_j) := \pi \circ \varepsilon(\underline{x} - \{x_m\} \to x_j)$ if $x_j < x_m$.[6] It can be seen that $\varepsilon(\underline{x} \hookrightarrow x_j)$ has the desired behavior: $(\mathsf{K}(\underline{c}), d) \in \mathcal{M}[\varepsilon(\underline{x} \hookrightarrow x_j)]$ iff $c_j = d$. Similarly, we can use fork and projections to extract the code of a subset. To extract the code of subset $\underline{x}$ from that of $\underline{x} \cup \{y\}$, when $y \notin \underline{x}$, we can define the *subset-extracting* formula $\varepsilon(\underline{x} \cup \{y\} \hookrightarrow \underline{x})$ by: $\varepsilon(\underline{x} \cup \{y\} \hookrightarrow \underline{x}) := \pi$ if $y > x_m$ and $\varepsilon(\underline{x} \cup \{y\} \hookrightarrow \underline{x}) := (\pi \circ \varepsilon(\underline{x} - \{x_m\} \cup \{y\} \hookrightarrow \underline{x} - \{x_m\})) \bigtriangledown \rho$ if $y < x_m$.[7] It can be seen that $\varepsilon(\underline{x} \cup \{y\} \hookrightarrow \underline{x})$ has the desired behavior: $(\mathsf{K}(\underline{c} \cup \{d\}), e) \in \mathcal{M}[\varepsilon(\underline{x} \cup \{y\} \hookrightarrow \underline{x})]$ iff $\mathsf{K}(\underline{c}) = e$.

5.3 General Translation from FOL to FAL

We will now indicate how to construct the general translation GT. For each first-order formula, we first classify its free variables as input and output[8] and then translate it to a fork formula with this behavior, modulo variable coding.

For atomic formulas, it suffices to provide translations for the cases mentioned in Lemma 3 on ordered term-reduced form.

An atomic formula $x \approx x$ is satisfied by any value of x. If we consider its variable x as both input and output, this input/output behavior can be described by $Id_{\mathcal{U}}$. Now, we have the fork formula $\iota\delta$ with this behavior. So, we set $GT(x \approx x) := \iota\delta$. An atomic formula $x \approx y$ with two distinct variables x and y is satisfied by equal values assigned to x and y. If we consider variables x and y, respectively, as input and output, the input/output behavior is $Id_{\mathcal{U}}$. We have the fork formula $\iota\delta$ with this behavior. So, we set $GT(x \approx y) := \iota\delta$. An atomic formula $x \star y \approx z$ with three distinct variables $x < y < z$ is satisfied by values c, d and e assigned to x, y and z, respectively, exactly when $c \star d = e$. If we consider variables x and y as input and z as output, the input variables will be coded by $\mathsf{K}(\{x, y\}) = x \star y$. Since we wish to compare $c \star d$ and e, this input/output behavior can be described by $Id_{\mathcal{U}}$. Thus, we also set $GT(x \star y \approx z) := \iota\delta$.

$$\underbrace{\begin{pmatrix} c \\ \star \\ d \end{pmatrix} \approx e}_{\mathcal{A}} \quad \Leftrightarrow \quad \underbrace{\begin{pmatrix} c \\ \star \\ d \end{pmatrix} \xrightarrow{\iota\delta} e}_{\mathcal{M}}$$

The remaining atomic formula to be considered is $P(x, y)$ with two distinct variables $x < y$. If we consider variables x and y, respectively, as input and output, the input/output behavior is given by $\mathcal{A}[P]$. Since $\mathcal{A}[P] = V(p)$, the arrow formula p has this behavior. So, we set $GT(P(x, y)) = p$.

[6] For instance, with $\underline{u} = \{x, y, z, w\}$, $\varepsilon(\underline{u} \hookrightarrow y)$ is $\pi \circ \pi \circ \rho$; so, $\varepsilon(\underline{u} \hookrightarrow y)$ on input $\mathsf{K}(\underline{u}) = ((x \star y) \star z) \star w$ will output y.

[7] For instance, with $\underline{v} = \{x, z, w\}$, $\varepsilon(\underline{v} \cup \{y\} \hookrightarrow \underline{v})$ is $((\pi \circ \pi \circ \pi) \bigtriangledown (\pi \circ \rho)) \bigtriangledown \rho$; so, $\varepsilon(\underline{v} \cup \{y\} \hookrightarrow \underline{v})$ on input $\mathsf{K}(\underline{v} \cup \{y\}) = ((x \star y) \star z) \star w$ will output $\mathsf{K}(\underline{v}) = (x \star z) \star w$.

[8] Given a first-order formula φ with its free variables classified as input and output variables, we will use $\mathsf{i}(\varphi)$ and $\mathsf{o}(\varphi)$ for the corresponding sets of variables.

The translation of non-atomic formulas is defined inductively. A negated formula $\neg\varphi$ has the same set of free variables as φ and we keep the same classification as input and output, say $\underline{x}$ and $\underline{y}$. So, the input/output behavior of $\neg\varphi$ consists exactly of those pairs $\langle \underline{a}, \underline{b} \rangle$ outside that of φ, i.e., $\mathcal{A}[\neg\varphi] = \overline{\mathcal{A}[\varphi]}$. We can thus set $GT(\neg\varphi) := \neg GT(\varphi)$.[9]

The translation of conjunction will be simplified if the formulas have the same free variables. To achieve this, we will introduce some auxiliary ideas.

Consider a first-order formula φ with free variables classified as input $\mathsf{i}(\varphi)$ and output $\mathsf{o}(\varphi)$. To add new variables we will employ conjunctions with equality. We use ${}_y\varphi$ for the *pre-conjunction with equality*, i.e., the formula $y{\approx}y \wedge \varphi$ with $\mathsf{i}({}_y\varphi) := \{y\} \cup \mathsf{i}(\varphi)$ and $\mathsf{o}({}_y\varphi) := \mathsf{o}(\varphi)$; dually, we use φ_y for the *post-conjunction with equality*: the formula $\varphi \wedge y{\approx}y$ with $\mathsf{i}(\varphi_y) := \mathsf{i}(\varphi)$ and $\mathsf{o}(\varphi_y) := \mathsf{o}(\varphi) \cup \{y\}$.

We now wish to translate such conjunctions with equality. If y is an input variable of formula φ, then ${}_y\varphi$ has the same input variables as φ; so we set $GT({}_y\varphi) := GT(\varphi)$. Similarly, if y is an output variable of formula φ, then we set $GT({}_y\varphi) := GT(\varphi)$. It remains to consider the cases when y is a new free variable.

Let us examine the case of ${}_y\varphi$ when y is a new input variable. Both formulas φ and ${}_y\varphi$ have the same output variables, but distinct inputs: if φ has input $\underline{w}$ coded by $K(\underline{w})$ then ${}_y\varphi$ will have input $\underline{w} \cup \{y\}$ coded by $K(\underline{w} \cup \{y\})$. To use the translation $GT(\varphi)$ for φ, we must obtain $K(\underline{w})$ from $K(\underline{w} \cup \{y\})$. But, we can extract $K(\underline{w})$ from $K(\underline{w} \cup \{y\})$ by the subset-extracting formula $\varepsilon(\underline{w} \cup \{y\} \hookrightarrow w)$. We can thus set $GT({}_y\varphi) := \varepsilon(\mathsf{i}(\varphi) \cup \{y\} \hookrightarrow \mathsf{i}(\varphi)) \circ GT(\varphi)$.[10]

$$\underbrace{\begin{array}{c} d \approx d \\ \wedge \\ \varphi(\underline{c}) \end{array}}_{\mathcal{A}} \quad \Leftrightarrow \quad \underbrace{K\left(\begin{array}{c}\{y\}\\ \cup \\ \underline{w}\end{array}\right) \xrightarrow{\varepsilon(\underline{w} \cup \{y\} \mapsto \underline{w})} K(\underline{w}) \xrightarrow{GT(\varphi)} K(\underline{z})}_{\mathcal{M}}$$

The case of φ_y when y is a new output variable is easily seen to be dual, using the converse of the subset-extracting fork formula $\varepsilon(\underline{z} \cup \{y\} \hookrightarrow z)$. We can thus set $GT(\varphi_y) := GT(\varphi) \circ (\otimes\varepsilon(\mathsf{o}(\varphi) \cup \{y\} \hookrightarrow \mathsf{o}(\varphi)))$.

$$\underbrace{\begin{array}{c} \varphi(\underline{c}) \\ \wedge \\ d \approx d \end{array}}_{\mathcal{A}} \quad \Leftrightarrow \quad \underbrace{K(\underline{w}) \xrightarrow{GT(\varphi)} K(\underline{z}) \xleftarrow{\varepsilon(\underline{z} \cup \{y\} \mapsto \underline{z})} K\left(\begin{array}{c}\underline{z}\\ \cup \\ \{y\}\end{array}\right)}_{\mathcal{M}}$$

In general, for a finite set $\underline{v} = \{v_1 \ldots, v_m\}$ of variables, we have the finite conjunctions with equality ${}_{\underline{v}}\varphi$ and $\varphi_{\underline{v}}$ with $\mathsf{i}({}_{\underline{v}}\varphi) := \underline{v} \cup \mathsf{i}(\varphi)$, $\mathsf{o}({}_{\underline{v}}\varphi) := \mathsf{o}(\varphi)$, $\mathsf{i}(\varphi_{\underline{v}}) := \mathsf{i}(\varphi)$, and $\mathsf{o}(\varphi_{\underline{v}}) := \mathsf{o}(\varphi) \cup \underline{v}$. We define the *pre-conjunction* ${}_{y \cup \underline{x}}\varphi :=$

[9] For instance, consider $P(x, y)$ with $\mathsf{i}(P(x, y)) = \{x\}$ and $\mathsf{o}(P(x, y)) = \{y\}$. Its negation has $\mathsf{i}(\neg P(x, y)) = \{x\}$ and $\mathsf{o}(\neg P(x, y)) = \{y\}$, whence, $GT(\neg P(x, y)) = \neg p$.

[10] For instance, consider $P(x, z)$ and a new variable y with $x < y < z$. Then, ${}_yP(x, z)$ is $y{\approx}y \wedge P(x, z)$ and $GT({}_yP(x, z)) = \pi \circ p$, since $GT(P(x, z)) = p$ and $\varepsilon(\mathsf{i}(\varphi) \cup \{y\} \hookrightarrow \mathsf{i}(\varphi)) = \varepsilon(\{x\} \cup \{y\} \hookrightarrow \{x\}) = \pi$.

${}_y({}_{\underline{x}}\varphi)$ and the *post-conjunction* $\varphi_{y\cup\underline{x}} := (\varphi_{\underline{x}})_y$. So, we can translate them by iterating the above translations.[11] We will employ ${}_{\underline{x}}\varphi_{\underline{y}}$ to abbreviate $({}_{\underline{x}}\varphi)_{\underline{y}}$, with $\mathsf{i}({}_{\underline{x}}\varphi_{\underline{y}}) := \underline{x} \cup \mathsf{i}(\varphi)$ and $\mathsf{o}({}_{\underline{x}}\varphi_{\underline{y}}) := \mathsf{o}(\varphi) \cup \underline{y}$.

With this apparatus, the translation of conjunction becomes simple: we set $GT(\varphi \wedge \theta) := GT({}_{\mathsf{i}(\theta)}\varphi_{\mathsf{o}(\theta)}) \wedge GT({}_{\mathsf{i}(\varphi)}\theta_{\mathsf{o}(\varphi)})$.[12]

$$\underbrace{\begin{array}{c}\varphi(\underline{a},\underline{b})\\ \wedge\\ \theta(\underline{c},\underline{d})\end{array}}_{\mathcal{A}} \quad\Leftrightarrow\quad \underbrace{K\begin{pmatrix}\underline{a}\\ \cup\\ \underline{c}\end{pmatrix} \begin{array}{c} GT(\underline{w}\approx\underline{w}\wedge\varphi\wedge\underline{z}\approx\underline{z}) \\ \longrightarrow \\ \cap \\ GT(\underline{u}\approx\underline{u}\wedge\theta\wedge\underline{v}\approx\underline{v}) \\ \longrightarrow \end{array} K\begin{pmatrix}\underline{b}\\ \cup\\ \underline{d}\end{pmatrix}}_{\mathcal{M}}$$

Finally, to translate an existential formula $\exists y\varphi$ (when variable y occurs free in φ), we resort to the ordered alphabetic variant φ^y introduced above: with $y' := \mathsf{nxt}(\mathsf{vr}(\varphi))$ and $\varphi' := \varphi(y/y')$ we will translate the alphabetic variant $\exists y'\,{}_{y'}\varphi'_{y'}$ logically equivalent to $\exists y\varphi$. We have $\mathsf{i}({}_{y'}\varphi'_{y'}) = \{y'\} \cup \mathsf{i}(\varphi')$ and $\mathsf{o}({}_{y'}\varphi'_{y'}) = \mathsf{o}(\varphi') \cup \{y'\}$. The desired behavior for $\exists y'({}_{y'}\varphi'_{y'})$ receives an input $\underline{c}$ and outputs $\underline{d}$ exactly when $\exists y'({}_{y'}\varphi'_{y'})$ is satisfied by $\underline{c}$, i.e., ${}_{y'}\varphi'_{y'}$ is satisfied by $\underline{c}$ and some $e \in \mathcal{U}$, which amounts to ${}_{y'}\varphi'_{y'}$ producing output $\underline{d} \star e$ on input $\underline{c} \star e$, for some $e \in \mathcal{U}$. Now, from the input $\underline{c}$ we can generate a companion e by applying the transposed left extractor $\pi_{\mathcal{U}}^{-1}$ to obtain some $\underline{c} \star e$, which we then feed to $GT({}_{y'}\varphi'_{y'})$ and use its output $\underline{d} \star e$ to obtain $\underline{d}$ by means of the left extractor $\pi_{\mathcal{U}}$. In short, $GT(\exists y'\,{}_{y'}\varphi'_{y'}) = \otimes\pi \circ GT({}_{y'}\varphi'_{y'}) \circ \pi$. We can thus set $GT(\exists y\varphi) := \otimes\pi \circ GT({}_{y'}\varphi^y_{y'}) \circ \pi$. [13]

$$\underbrace{\exists y\varphi\begin{pmatrix}\underline{c},\underline{d}\\ ,\\ y\end{pmatrix}}_{\mathcal{A}} \quad\Leftrightarrow\quad \underbrace{\underline{c} \;\overset{\pi}{\longleftarrow}\; \begin{pmatrix}\underline{c}\\ \star\\ e\end{pmatrix} \;\xrightarrow{\;T({}_{y'}\varphi'_{y'})\;}\; \begin{pmatrix}\underline{d}\\ \star\\ e\end{pmatrix} \;\overset{\pi}{\longrightarrow}\; \underline{d}}_{\mathcal{M}}$$

[11] For instance, consider formula $P(x,z)$ with $\mathsf{i}(P(x,z)) = \{x\}$ and $\mathsf{o}(P(x,z)) = \{z\}$, as well as new variables y and w with $x < y < z < w$. We then have ${}_{y,w}P(x,z) = {}_w({}_yP(x,z)) = {}_w(y{\approx}y \wedge P(x,z)) = w{\approx}w \wedge y{\approx}y \wedge P(x,z)$, with $\mathsf{i}({}_{y,w}P(x,z)) = \{x,y,w\}$ and $\mathsf{o}({}_{y,w}P(x,z)) = \{z\}$. Now, since $GT({}_yP(x,z)) = \pi \circ p$, $GT({}_{y,w}P(x,z)) = \pi \circ (\pi \circ p)$.

[12] For instance, for formulas $P(u,v)$ (with $\mathsf{i}(P(u,v)) = \{u\}$ and $\mathsf{o}(P(u,v)) = \{v\}$), and $x \star y{\approx}z$ (with $\mathsf{i}(x \star y{\approx}z) = \{x,y\}$ and $\mathsf{o}(x \star y{\approx}z) = \{z\}$): $GT(P(u,v) \wedge x \star y{\approx}z) = GT(x{\approx}x \wedge y{\approx}y \wedge P(u,v) \wedge z{\approx}z) \wedge\ GT(u{\approx}u \wedge x \star y{\approx}z \wedge\ v{\approx}v)$.

[13] For instance, consider $\exists y(P(u,y) \wedge z \star w{\approx}u)$ with $u < v < x < y < z < w$ and no variable between z and w. Then, $y' = \mathsf{nxt}(\mathsf{vr}(P(u,v) \wedge x \star y{\approx}z)) = \mathsf{nxt}(u,v,x,y,z) = w$ and $(P(u,v) \wedge x \star y{\approx}z)' = (P(u,v) \wedge x \star y{\approx}z)(y/y') = P(u,v) \wedge x \star w{\approx}z$, whence ${}_{y'}(P(u,v) \wedge x \star y{\approx}z)'_{y'} = {}_w(P(u,v) \wedge x \star y{\approx}z)'_w = w{\approx}w \wedge (P(u,v) \wedge x \star w{\approx}z) \wedge w{\approx}w$, with $\mathsf{i}({}_w(P(u,v) \wedge x \star y{\approx}z)'_w) = \{u,x,y,w\}$ and $\mathsf{o}({}_w(P(u,v) \wedge x \star y{\approx}z)'_w) = \{v,z,w\}$. The ordered alphabetic variant is $\exists w({}_w(P(u,v) \wedge x \star y{\approx}z)'_w)$, i.e., $\exists w(w{\approx}w \wedge (P(u,v) \wedge x \star w{\approx}z) \wedge w{\approx}w)$; thus, as $GT(\exists y(P(u,v) \wedge x \star y{\approx}z)) = \otimes\pi \circ GT({}_{y'}(P(u,v) \wedge x \star y{\approx}z)'_{y'}) \circ \pi$, we have $GT(\exists y(P(u,v) \wedge x \star y{\approx}z)) = \otimes\pi \circ GT(w{\approx}w \wedge (P(u,v) \wedge x \star w{\approx}z) \wedge w{\approx}w) \circ \pi$.

Table 1 summarizes our input/output conventions.

Table 1. Input-output conventions

Formula φ	input set $\mathsf{i}(\varphi)$	output set $\mathsf{o}(\varphi)$	
$x{\approx}x$	$\{x\}$	$\{x\}$	
$x{\approx}y$	$\{x\}$	$\{y\}$	
$x \star y{\approx}z$	$\{x, y\}$	$\{z\}$	
$P(x, y)$	$\{x\}$	$\{y\}$	
$\neg\varphi$	$\mathsf{i}(\varphi)$	$\mathsf{o}(\varphi)$	
${}_y\varphi$	$\{y\} \cup \mathsf{i}(\varphi)$	$\mathsf{o}(\varphi)$	
φ_y	$\mathsf{i}(\varphi)$	$\mathsf{o}(\varphi) \cup \{y\}$	
$\underline{x}\varphi_{\underline{y}}$	$\underline{x} \cup \mathsf{i}(\varphi)$	$\mathsf{o}(\varphi) \cup \underline{y}$	
$\varphi \wedge \psi$	$\mathsf{i}(\varphi) \cup \mathsf{i}(\psi)$	$\mathsf{o}(\varphi) \cup \mathsf{o}(\psi)$	
$\exists y\varphi$	$\mathsf{i}(\varphi)$	$\mathsf{o}(\varphi)$	when $y \notin \mathsf{fv}(\varphi)$
$\exists y\varphi$	$\mathsf{i}(\varphi) - \{y\}$	$\mathsf{o}(\varphi) - \{y\}$	when $y \in \mathsf{fv}(\varphi)$

Table 2 summarizes our general translation.

Table 2. GT translation

for any x:	$GT(x{\approx}x) := \iota\delta$
for any distinct x and y:	$GT(x{\approx}y) := \iota\delta$
for any distinct $x < y < z$:	$GT(x \star y{\approx}z) := \iota\delta$
for any distinct $x < y$:	$GT(P(x, y)) := p$
for any φ:	$GT(\neg\varphi) := \neg GT(\varphi)$
for any φ and y:	$GT({}_y\varphi) := GT(\varphi)$ if $y \in \mathsf{i}(\varphi)$
	$GT(\varphi_y) := GT(\varphi)$ if $y \in \mathsf{o}(\varphi)$
for any φ and y:	$GT({}_y\varphi) := \varepsilon(\{y\} \cup \mathsf{i}(\varphi) \hookrightarrow \mathsf{i}(\varphi)) \circ GT(\varphi)$
	if $y \notin \mathsf{i}(\varphi)$
	$GT(\varphi_y) := GT(\varphi) \circ (\otimes\varepsilon(\mathsf{o}(\varphi) \cup \{y\} \hookrightarrow \mathsf{o}(\varphi)))$
	if $y \notin \mathsf{o}(\varphi)$
for any φ and ψ:	$GT(\varphi \wedge \psi) := GT({}_{\mathsf{i}(\psi)}\varphi_{\mathsf{o}(\psi)}) \wedge GT({}_{\mathsf{i}(\varphi)}\psi_{\mathsf{o}(\varphi)})$
for φ and $y \notin \mathsf{fv}(\varphi)$:	$GT(\exists y\varphi) := GT(\varphi)$
for φ and $y \in \mathsf{fv}(\varphi)$:	$GT(\exists y\varphi) := (\otimes\pi) \circ GT({}_{y'}\varphi^{y}_{y'}) \circ \pi$

Lemma 4 (Behavior of GT on similar models). *Given a first-order formula φ of FOL$[\underline{\mathsf{P}}, \star]$ with $m > 0$ free variables classified as input $\mathsf{i}(\varphi) = \underline{x}$ and output $\mathsf{o}(\varphi) := \underline{y}$, consider similar models $\mathcal{M}$ for* FAL$[\underline{\mathsf{P}}]$ *and $\mathcal{A}$ for* FOL$[\underline{\mathsf{P}}, \star]$. *Then whenever $\mathcal{A} \models K(\underline{x}){\approx}z[\underline{a}, c]$ and $\mathcal{A} \models K(\underline{y}){\approx}w[\underline{b}, d]$, we have $\mathcal{A} \models \varphi(\underline{x}, \underline{y})[\underline{a}, \underline{b}]$ iff $(b, d) \in \mathcal{M}[GT(\varphi)]$.*

Proof. By the inductive construction of $GT(\varphi)$, we have $\mathcal{A} \models \varphi(\underline{x}, \underline{y})[\underline{a}, \underline{b}]$ iff $(\mathsf{K}(\underline{a}), \mathsf{K}(\underline{b})) \in [GT(\varphi)]$. □

Given a model $\mathcal{A}$ for FOL[$\underline{\mathsf{P}}, \star$] and a formula φ of FOL[$\underline{\mathsf{P}}, \star$] with m free variables classified as input $\mathsf{i}(\varphi) = \underline{x}$ and output $\mathsf{o}(\varphi) = \underline{y}$, we now define the following behaviors of formula φ in model $\mathcal{A}$:
input/output behavior: $\mathcal{A}_{\mapsto}[\varphi] := \{(\underline{a}, \underline{b}) \in \mathcal{U}^m : \mathcal{A} \models \varphi(\underline{x}, \underline{y})[\underline{a}, \underline{b}]\}$;
coded input/output behavior: $\mathcal{A}_{\mathsf{K}}[\varphi] := \{(\mathsf{K}(\underline{a}), \mathsf{K}(\underline{b})) \in \mathcal{U}^2 : (\underline{a}, \underline{b}) \in \mathcal{A}_{\mapsto}[\varphi]\}$.

Proposition (Coded behavior of GT on similar models). *Consider similar models $\mathcal{M}$ for* FAL[P] *and $\mathcal{A}$ for* FOL[$\underline{\mathsf{P}}, \star$] *and formula φ of* FOL[$\underline{\mathsf{P}}, \star$] *(with at least one free variable). The coded input/output behavior of φ in $\mathcal{A}$ is the behavior of $GT(\varphi)$, i.e., $\mathcal{A}_{\mathsf{K}}[\varphi] = \mathcal{M}[GT(\varphi)]$.*

Proof. By Lemma 4 on behavior of GT on similar models: $(\underline{a}, \underline{b}) \in \mathcal{A}_{\mapsto}[\varphi]$ iff $(\mathsf{K}(\underline{a}), \mathsf{K}(\underline{b}) \in \mathcal{M}[GT(\varphi)]$. □

These results can also be stated entirely within FOL in terms of provable behavior of the general and square translations[14].

5.4 Reverse Translation from FOL to FAL

We can now define the reverse translation RT_{xy} from formulas of FOL[$\underline{\mathsf{P}}, \star$] with exactly two free variables to formulas of FAL[P] and show that it achieves its aim: both formulas have the same behavior. Consider formula φ of FOL[$\underline{\mathsf{P}}, \star$] with two distinct free variables x and y. As such, its extension in a model $\mathcal{A}$ for FOL[$\underline{\mathsf{P}}, \star$] is a binary relation: $\mathcal{A}[\varphi] := \{(a, b) \in \mathcal{U}^2 : \mathcal{A} \models \varphi[a, b]\}$. We thus have an input/output behavior, if we classify variables x and y as input and output.

The *reverse translation* RT_{xy} from formulas of FOL[$\underline{\mathsf{P}}, \star$] with exactly two free variables to formulas of FAL[P] is defined by $RT_{xy}(\varphi) := GT(\varphi)$, for any first-order formula φ with two free variables x and y.

Theorem 3 (Behavior of RT on similar models). *Given similar models $\mathcal{M}$ for* FAL[P] *and $\mathcal{A}$ for* FOL[$\underline{\mathsf{P}}, \star$] *and formula φ of* FOL[$\underline{\mathsf{P}}, \star$] *with free variables x and y classified as input and output, respectively, the input/output behavior of φ in $\mathcal{A}$ is the same of the reverse translation $RT_{xy}(\varphi)$ of φ in $\mathcal{M}$: $\mathcal{A}_{\mapsto}[\varphi] = \mathcal{M}[RT_{xy}(\varphi)]$.*

Proof. By the proposition on behavior of general translation on similar models, since $K(\{x\}) = x$ and $K(\{y\}) = y$. □

5.5 Forward and Backward Translations

We have forward translations ST_{xy} from fork formulas to first-order formulas, as well as backward translations GT and RT_{xy} from first-order formulas to fork formulas. We can compose these forward and backward translations in two ways. The next result shows that either composition maps a formula to one with the same behavior.

[14] Indeed, for a formula φ of FOL[$\underline{\mathsf{P}}, \star$] with $m > 0$ free variables classified as input and output: Injective($\star$) $\vdash_{\mathrm{FOL}}$ $\mathsf{K}(\mathsf{i}(\varphi))\approx x \wedge \mathsf{K}(\mathsf{o}(\varphi))\approx y) \rightarrow (\varphi \leftrightarrow ST_{xy}(GT(\varphi))$.

Theorem 4 (Behavior of composite translations).
i) Given a model $\mathcal{M}$ for FAL[P]*, for each formula α of* FAL[P] *we have $\mathcal{M}[\alpha] = \mathcal{M}[RT_{xy}(ST_{xy}(\alpha))]$.*
ii) Given a model $\mathcal{A}$ for FOL[$\underline{\mathsf{P}}, \star$]*, for each formula φ of* FOL[$\underline{\mathsf{P}}, \star$] *with two distinct free variables x and y classified as input and output, respectively, we have $\mathcal{A}[\varphi] = \mathcal{A}[ST_{xy}(RT_{xy}(\varphi))]$.*

Proof. By Theorem 3 on behavior of RT_{xy} on similar models and Theorem 2 on similar models and ST_{xy}. □

The next result shows that either composition of translations maps a formula to an equivalent one.

Corollary (Provable behavior of composite translations).
i) Given formula α of FAL[P]*, we have $\vdash_{\mathrm{FAL}} \alpha \leftrightarrow RT_{xy}(ST_{xy}(\alpha))$.*
ii) Given formula φ of FOL[$\underline{\mathsf{P}}, \star$] *with two distinct free variables x and y classified as input and output, respectively,* Injective($\star$) $\vdash_{\mathrm{FOL}} \varphi \leftrightarrow ST_{xy}(RT_{xy}(\varphi))$.

Proof. By Theorem 4 on behavior of composite translations on similar models (since for each first-order model $\mathcal{A}$ for Injective($\star$), we have a similar fork square model $\mathcal{M}$). □

6 Perspective

This work continues the investigation of fork arrow logic started in [BV99]. The development of this logic can be seen in the more general context of Brink-Venema's diagram (Figure 1 in Section 1).

Fork arrow logic is defined from fork algebras much as arrow logic was defined from relation algebras. In [FVVVB] we provide an orthodox finite axiomatization of fork squares: the square frames for fork arrow logic.

In this paper we provide a partial answer to questions addressed in [FVVVB], namely, does fork arrow logic have the expressive power of full first-order logic?

The behavior of a formula of fork arrow logic can be defined as its extension in a square model, i.e the input/output behavior expressed by the modal formula is the set of pairs (binary relation) where the formula is satisfied in the model. Similarly, the behavior of a first-order formula with two free variables can be defined as its extension in a first-order structure, i.e., the input/output behavior expressed by the first-order formula is the set of assingments to the free variables that satisfy the formula (a binary relation on the universe of the first-order structure). For each pair of variables x and y, we provide two translations, one from first-order formulas with free variables x and y to fork formulas and another one in the opposite direction. We show that they are inverse to each other up to equivalence of formulas. Thus, we know that every input/output behavior that one can express in first-order logic can also be expressed in fork arrow logic.

The apparatus of arrow logic provides means for specifying some behaviors of processes obtained by combining behaviors of component processes, though not

contemplating some combined behaviors, such as process creation, parallelism and synchronization. Being so, the addition of fork to arrow logic enriches its expressive power for computational processes. We can also hope that fork arrow logic inherits from fork algebra the usefulness in reasoning about specifications and derivation of programs.

It should also be noted that reasoning in a modal context with the square semantics seems to be very advantageous: one deals only with binary relations (in contrast with n-ary relations for arbitrary n) and with squares (simple algebras) at the meta level. So, the correspondence language involves only equality and binary predicates in addition to a unary function symbol. The expressivity correspondence between first-order logic and fork arrow logic established here can be advantageously used: one can freely choose either formalism to express and reason about input/output behavior, so one has transfer of applicability and properties between both formalisms.

In addition, it seems natural to express and to reason about input/output behaviors using a language with binary predicates. We plan to continue investigating the application of fork arrow logic to describe program behavior along lines similar to Maddux's approach [Mad96]. Also, arrow logic has been used to formalize notions in a wide spectrum of areas: mathematics, computer science, linguistics, cognitive science. We pointed out how fork arrow logic enriches the applicability of arrow logic in computer science. We expect that one can have the same improvement with respect to other areas of applications using fork arrow logic.

Acknowledgements

The authors gratefully acknowledge partial financial support from the Brazilian National Research Council (CNPq).

References

[vBen84] J. van Benthem, Correspondence theory. In D. M. Gabbay and F. Guenthner (eds.), *Handbook of Philosophical Logic*, vol. 2, Reidel, Dordrecht, 1984, 167–248.

[vBen91] J. van benthem, *Language in Action: Categories, Lambdas and Dynamic Logic*. North-Holland, Amsterdam, 1991.

[vBen94] J. van benthem, A note on dynamic arrow logic. In J. van Eijck and A. Visser (eds.), *Logic and Information Flow*, MIT Press, Cambridge, 1994.

[BV99] M.R.F. Benevides and P.A.S. Veloso. Axiomatization and completeness for fork modal logic. In *XII Encontro Brasileiro de Lógica*, 25 a 28 de maio, Itatiaia, 87–94, 1999.

[EFT84] H.-D. Ebbinghaus, J. Flum, and W. Thomas, *Mathematical Logic*, Springer, Berlin, 1984.

[FBH97] M.F. Frias, G.A. Baum and A.M. Haeberer, Fork algebras in algebras, logic and computer science, *Fundamenta Informaticæ*, **32**(1997) 1–25.

[FVVVB] R.P. de Freitas, J.P. Viana, P.A.S. Veloso, S.R.M. Veloso, and M.R.F. Benevides, Squares in fork arrow logic, *Journal of Philosophical Logic*, to appear.

[HFBV97] A.M. Haeberer, M.F. Frias, G.A. Baum, and P.A.S. Veloso, Fork algebras. In C. Brink, W. Kahl and G. Schmidt (eds.), *Relational Methods in Computer Science*, Springer, Vienna, 1997, 54–69.

[Jon82] B. Jónsson, Varieties of relation algebras, *Algebra Universalis*, **15** (1982), 273–298.

[Kri59] S. Kripke, A completeness theorem in modal logic, *The Journal of Symbolic Logic*, **24**(1959) 1–14.

[Lem66] E.J. Lemmon, Algebraic semantics for modal logic I and II, *The Journal of Symbolic Logic* **31**(1966) 46–65 and 191–218.

[Mad96] R.D. Maddux, Relation-algebraic semantics, *Theoretical Computer Science* **160**(1996) 1–85.

[Mar95] M. Marx, *Relativization in Arrow Logic*, Doctoral dissertation, Institute for Logic, Language and Computation, Universiteit van Amsterdan, 1995.

[MPM96] M. Marx, L. Pólos and M. Masuch, *Arrow Logic and Multi-Modal Logic*, CSLI Publications, Stanford, 1996.

[MV97] M. Marx and Y. Venema, *Multi-Dimensional Modal Logic*, Applied Logic Series, vol. 4, Kluwer Academic Publishers, Dordrecht, 1997.

[Nem91] I. Neméti, Algebraizations of quantifier logics, an introductory overview, *Studia Logica* L(3/4), **50**(1991) 485–569.

[Vel99] P.A.S. Veloso, Some connections between Logic and Computer Science. In W. Carnielli and I.M.L d'Ottaviano (eds.), *Advances in Contemporary Logic and Computer Science*, AMS, Providence, 1999, 187–260.

[Ven91] Y. Venema, *Many-Dimensional Modal Logic*, Doctoral dissertation, Institute for Logic, Language and Computation, Univ. Amsterdan, 1991.

[Ven96] Y. Venema, A Crash Course in Arrow Logic. In [MPM96] 3–34.