Executable First-Order Queries in the Logic of Information Flows *

Non Peer-reviewed author version

Aamer, Heba A.; Bogaerts , Bart; SURINX, Dimitri; Ternovska, Eugenia & VAN DEN BUSSCHE, Jan (2024) Executable First-Order Queries in the Logic of Information Flows *. In: Logical Methods in Computer Science, 20 (2).

DOI: 10.46298/LMCS-20(2:6)2024
Handle: http://hdl.handle.net/1942/43086

Logical Methods in Computer Science
Volume 20, Issue 2, 2024, pp. 6:1–6:41
https://lmcs.episciences.org/

Submitted Oct. 05, 2022
Published May 08, 2024

EXECUTABLE FIRST-ORDER QUERIES IN THE LOGIC OF INFORMATION FLOWS *

HEBA AAMER [a], BART BOGAERTS [a], DIMITRI SURINX [b], EUGENIA TERNOVSKA [c], AND JAN VAN DEN BUSSCHE [b]

[a] Vrije Universiteit Brussel, Belgium
e-mail address: heba.mohamed@vub.be,bart.bogaerts@vub.be

[b] Hasselt University, Belgium
e-mail address: surinxd@gmail.com,jan.vandenbussche@uhasselt.be

[c] Simon Fraser University, Canada
e-mail address: ter@sfu.ca

Abstract. The logic of information flows (LIF) has recently been proposed as a general framework in the field of knowledge representation. In this framework, tasks of procedural nature can still be modeled in a declarative, logic-based fashion. In this paper, we focus on the task of query processing under limited access patterns, a well-studied problem in the database literature. We show that LIF is well-suited for modeling this task. Toward this goal, we introduce a variant of LIF called "forward" LIF (FLIF), in a first-order setting. FLIF takes a novel graph-navigational approach; it is an XPath-like language that nevertheless turns out to be equivalent to the "executable" fragment of first-order logic defined by Nash and Ludäscher. One can also classify the variables in FLIF expressions as inputs and outputs. Expressions where inputs and outputs are disjoint, referred to as io-disjoint FLIF expressions, allow a particularly transparent translation into algebraic query plans that respect the access limitations. Finally, we show that general FLIF expressions can always be put into io-disjoint form.

Introduction

An information source is said to have a limited access pattern if it can only be accessed by providing values for a specified subset of the attributes; the source will then respond with tuples giving values for the remaining attributes. A typical example is a restricted telephone directory D(name; tel) that will show the phone numbers for a given name, but not the other way around. For another example, the public bus company may provide its weekdays schedule as a relation Route(stop, interval; time, line, next, duration) that, given a

Key words and phrases: Limited access pattern,expressive power,variable substitution,composition.

* This paper is the combined, extended, and fully revised journal version of two papers presented at ICDT 2020 and ICDT 2021 [ABS+20, AVdB21].

This work was partially supported by FWO project G0D9616N and by the Flanders AI Research Program. Heba Aamer was supported by the Special Research Fund (BOF) (BOF19OWB16) while at Hasselt University. Jan Van den Bussche is partially supported by the National Natural Science Foundations of China (61972455).

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.46298/LMCS-20(2:6)2024

bus stop and a time interval, outputs bus lines that stop there at a time within the interval, together with the duration to the next stop. Note how we use a semicolon to separate the attributes required to access the information source from the rest of the attributes.

The topic of querying information sources with limited access patterns was put on the research agenda in the mid 1990s [RSU95], and has been intensively investigated since then, with recent work until at least 2018 [YLGMU99, FLMS99, DGL00, Li03, MHF03, NL04, DLN07, CM08b, CM08a, CCM09, BGS11, BBB13, BLT15, BtCT16, CMRU17, CU18]. The research is motivated by diverse applications, such as query processing using indices, information integration, or querying the Deep Web. A review of the field was given by Benedikt et al. [BLtCT16, Chapter 3.12].

In this paper, we offer a fresh perspective on querying with limited access patterns, based on the Logic of Information Flows (LIF). This framework has been recently introduced in the field of knowledge representation [Ter17, Ter19]. The general aim of LIF is to model how information propagates in complex systems. LIF allows machine-independent characterizations of computation; in particular, it allows tasks of a procedural nature to be modeled in a declarative fashion.

In the full setting, LIF is a rich family of logics with higher-order features. The present paper is self-contained, however, and we introduce here a lightweight, first-order fragment of LIF, which we call *forward* LIF (FLIF). Our goal then is to show that FLIF is suitable to query information sources with limited access patterns.

Specifically, we offer the following insights and contributions:

(1) We offer a new perspective on databases with access limitations, by viewing them as a graph. The nodes of the graph are valuations; the edges denote access to information sources. The start node of an edge provides values to input variables, and the end node provides values to output variables.
(2) Our perspective opens the door to using a graph query language to query databases with access limitations. Standard navigational graph query languages [PAG10, FGL$^+$15, LMV13, SFG$^+$15, AAB$^+$17] have a logical foundation in Tarski's algebra of binary relations [Tar41, Mad91, Pra92, tCM07]. However, in our situation, nodes in a graph are not abstract elements, but valuations that give values to variables.
(3) Interestingly, LIF, in its first-order version, can be understood exactly as the desired extension of Tarski's algebra to binary relations of valuations. LIF is a *dynamic* logic: like first-order dynamic logic [HKT00] or dynamic predicate logic [GS91], expressions of LIF are not satisfied by single valuations, but by *pairs* of valuations. Such pairs represent transitions of information. However, LIF is very general and has operators, such as converse, or cylindrification, which do not rhyme with the limited access to information sources that we want to target in this work. Therefore, in this paper, we introduce FLIF, an instantiation of the LIF framework where information can only flow forward. Like navigational graph query languages, FLIF expressions define sets of pairs of valuations so that there is a path in the graph from the first valuation of the pair to the second.
(4) We show that FLIF is equivalent in expressive power to executable FO, an elegant syntactic fragment of first-order logic introduced by Nash and Ludäscher [NL04]. Formulas of executable FO can be evaluated over information sources in such a way that the limited access patterns are respected. Furthermore, the syntactical restrictions are not very severe and become looser the more free variables are declared as inputs.

(5) Our equivalence result between FLIF and executable FO is interesting since FLIF is a simple compositional language, built from atomic expressions using just three navigational operators: composition, union, and difference. These operators allow one to build paths, explore alternatives, and exclude paths. The atomic expressions are information accesses, tests, or variable assignments. Thus, FLIF is a very different language from executable FO, where the classical first-order constructs (disjunction, conjunction, negation, quantification) are syntactically restricted to be ordered so as to respect the access limitations, and cannot simply be combined orthogonally. FLIF, which directly navigates through the graph, is also different from other approaches in the literature where first the "accessible part" (up to some depth) of the database is retrieved, after which an arbitrary query can be evaluated on this part.
(6) We also specialize our result to FLIF expressions that are *io-disjoint*. This is a property coming from our companion paper where we analyze input and output sensitivity in LIF expressions [ABS$^+$23]. An expression α is io-disjoint if, whenever α can reach a valuation ν_{out} from a valuation ν_{in}, the values of the variables in ν_{out} depend only on the values of variables in ν_{in} that have not changed in ν_{out}. For io-disjoint expressions, the single valuation ν_{out} contains all the relevant information: in this sense, the io-disjoint fragment of FLIF can be given a static (single-valuation) semantics as opposed to the dynamic semantics of full FLIF.
(7) We show three results on io-disjoint FLIF. First, when translating FLIF to executable FO, a more economical translation is possible if the FLIF expression is io-disjoint. Here, by "economical", we mean that fewer variables are needed in the FO formula, and the FO formula is closer in syntax to the FLIF expression.
(8) Second, we show that io-disjoint FLIF expressions can be translated into *plans* in a particularly simple and transparent manner. Plans are a standard way of formalizing query processing with limited access patterns [BLtCT16]. In such plans, database relations can only be accessed by joining them on their input attributes with a relation that is either given as input or has already been computed. Apart from that, plans can use the usual relational algebra operations. That executable FO can be translated into plans is well known, so, by the equivalence with FLIF, the same holds for FLIF. However, the resulting plans can be rather complex, just like the classical translation from relational calculus to relational algebra [AHV95] can produce rather ugly algebra expressions in general. So, our result is that for io-disjoint FLIF, very simple plans can be produced. The plans we generate do not need the renaming operator, and use only natural joins (no cartesian products or theta-joins).
(9) Third, we show that, actually, any FLIF expression can be simulated by an io-disjoint one. The simulation requires auxiliary variables and variable renamings, and the correctness proof is quite intricate. We see this result mainly as an expressiveness result, not as suggesting a practical way to evaluate arbitrary FLIF expressions. Indeed, these can be evaluated rather directly as is, since FLIF is an algebraic language in itself.

This paper is further organized as follows. We begin with some preliminaries in Section 1. Section 2 introduces the language FLIF. In Section 3, we recall the basic setting of executable FO on databases with limited access patterns; furthermore, we prove the equivalence between FLIF and executable FO. In Section 4, we formally define the io-disjoint fragment. Then, in Section 5, we give a translation from that fragment to executable FO which improves upon the translation from FLIF from Section 3. In Section 5, we also give a translation from FLIF to its io-disjoint fragment. In Section 6, we give the correctness proofs of the

translation theorems from Sections 3 and 5. Section 7 discusses evaluation plans. Finally, we discuss related work and then conclude in Sections 8 and 9 respectively.

1. Preliminaries

Relational database schemas are commonly formalized as finite relational vocabularies, i.e., finite collections of relation names, each name with an associated arity (a natural number). To model limited access patterns, we additionally specify an *input arity* for each name. For example, if R has arity five and input arity two, this means that we can only access R by giving input values, say a_1 and a_2, for the first two arguments; R will then respond with all tuples $(x_1, x_2, x_3, x_4, x_5)$ in R where $x_1 = a_1$ and $x_2 = a_2$.

Thus, formally, we define a *database schema* as a triple $\mathcal{S} = (\mathit{Names}, \mathit{ar}, \mathit{iar})$, where *Names* is a set of relation names; *ar* assigns a natural number $\mathit{ar}(R)$ to each name R in *Names*, called the arity of R; and *iar* similarly assigns an input arity to each R, such that $\mathit{iar}(R) \leq \mathit{ar}(R)$. In what follows, we use $\mathit{oar}(M)$ (output arity) for $\mathit{ar}(M) - \mathit{iar}(M)$.

Remark 1.1. In the literature, a more general notion of schema is often used, allowing, for each relation name, several possible sets of input arguments; each such set is called an access method. In this paper, we stick to the simplest setting where there is only one access method per relation, consisting of the first k arguments, where k is set by the input arity. All subtleties and difficulties already show up in this setting. Nevertheless, our definitions and results can be easily generalized to the setting with multiple access methods per relation.

The notion of database instance remains the standard one. Formally, we fix a countably infinite universe **dom** of atomic data elements, also called *constants*. Now an *instance* D of a schema $\mathcal{S}$ assigns to each relation name R an $\mathit{ar}(R)$-ary relation $D(R)$ on **dom**. We say that D is *finite* if every relation $D(R)$ is finite. The *active domain* of D, denoted by $\mathrm{adom}(D)$, is the set of all constants appearing in the relations of D.

The syntax and semantics of first-order logic (FO, relational calculus) over $\mathcal{S}$ is well known [AHV95]. The set of free variables of an FO formula φ is denoted by $\mathit{fvars}(\varphi)$. Moreover, in formulas, we allow constants only in equalities of the form $x = c$, where x is a variable and c is a constant. As we mentioned earlier, in writing relation atoms, we find it clearer to separate input arguments from output arguments by a semicolon. Thus, we write relation atoms in the form $R(\bar{x}; \bar{y})$, where $\bar{x}$ and $\bar{y}$ are tuples of variables such that the length of $\bar{x}$ is $\mathit{iar}(R)$ and the length of $\bar{y}$ is $\mathit{oar}(R)$. For example, the relation atom $R(x, z; y, y, z)$ indicates that R is a relation name with $\mathit{ar}(R) = 5$ and $\mathit{iar}(R) = 2$; consequently, $\mathit{oar}(R) = 3$.

We use the "natural" semantics [AHV95] and let variables in formulas range over the whole of **dom**. Formally, an *X-valuation* is a valuation defined on a set X of variables, and precisely, it is a mapping $\nu : X \to \mathbf{dom}$. We will often not specify the set X of variables a valuation is defined on when it is clear from context. It is convenient to be able to apply valuations also to constants, agreeing that $\nu(c) = c$ for any valuation ν and any $c \in \mathbf{dom}$. Moreover, in general, for a valuation ν, a variable x, and a constant c, we use $\nu[x := c]$ for the valuation that is the same as ν except that x is mapped to c. Additionally, we say that two valuations ν_1 and ν_2 agree on (outside) a set of variables X when $\nu_1(x) = \nu_2(x)$ for every variable $x \in X$ ($x \notin X$). Finally, given an instance D of $\mathcal{S}$, an FO formula φ over $\mathcal{S}$, and a valuation ν defined on $\mathit{fvars}(\varphi)$, the definition of when φ is satisfied by D and ν, denoted by $D, \nu \models \varphi$, is standard.

2. Forward LIF

In this section, we introduce the language FLIF.[1] The language itself is a form of dynamic logic. Indeed, the semantics of any FLIF expression is defined as a set of *pairs* of valuations. The operators are an algebraization of first-order logic connectives. Although FLIF is a dynamic algebraic form of first-order logic, it is notable that it lacks quantification operators, which makes it especially simple.

Syntax and semantics of FLIF: atomic expressions. The central idea is to view a database instance as a graph. The nodes of the graph are all possible valuations on some set of variables (hence the graph is infinite.) The edges in the graph are labeled with *atomic FLIF expressions*. Some of the edges are merely tests (i.e., self-loops), while other edges represent a change in the state.

Syntactically, over a schema $\mathcal{S}$ and a set of variables $\mathbb{V}$, there are five kinds of atomic expressions τ, given by the following grammar:

$$\tau ::= R(\bar{x};\bar{y}) \mid (x = y) \mid (x = c) \mid (x := y) \mid (x := c)$$

Here, $R(\bar{x};\bar{y})$ is a relation atom over $\mathcal{S}$ as in first-order logic with $\bar{x}$ and $\bar{y}$ being tuples of variables in $\mathbb{V}$, x and y are variables from $\mathbb{V}$, and c is a constant. The atomic expressions $(x = y)$ and $(x = c)$ are equality tests, while the expressions $(x := y)$ and $(x := c)$ are assignment expressions. From the grammar, we see that any atomic expression τ is defined such that $\mathit{vars}(\tau) \subseteq \mathbb{V}$ where $\mathit{vars}(\tau)$ is the set of variables used in τ.

Semantically, given an instance D of $\mathcal{S}$, a set of variables $\mathbb{V}$, and an atomic expression τ over $\mathcal{S}$ and $\mathbb{V}$, we define the set of τ-labeled edges in the graph view of D as a set $[\![\tau]\!]^{\mathbb{V}}_D$ of ordered pairs of $\mathbb{V}$-valuations, as follows.

Definition 2.1.

(1) $[\![R(\bar{x};\bar{y})]\!]^{\mathbb{V}}_D$ is the set of all pairs (ν_1, ν_2) of $\mathbb{V}$-valuations such that the concatenation $\nu_1(\bar{x}) \cdot \nu_2(\bar{y})$ belongs to $D(R)$, and ν_1 and ν_2 agree outside the variables in $\bar{y}$.
(2) $[\![(x = y)]\!]^{\mathbb{V}}_D$ is the set of all identical pairs (ν, ν) of $\mathbb{V}$-valuation such that $\nu(x) = \nu(y)$.
(3) Likewise, $[\![(x = c)]\!]^{\mathbb{V}}_D$ is the set of all identical pairs (ν, ν) of $\mathbb{V}$-valuation such that $\nu(x) = c$.
(4) $[\![(x := y)]\!]^{\mathbb{V}}_D$ is the set of all pairs (ν_1, ν_2) of $\mathbb{V}$-valuations such that $\nu_2 = \nu_1[x := \nu_1(y)]$. Thus, $\nu_2(x) = \nu_1(y)$ and ν_2 agrees with ν_1 on all other variables.
(5) Similarly, $[\![(x := c)]\!]^{\mathbb{V}}_D$ is the set of all pairs (ν_1, ν_2) of $\mathbb{V}$-valuations such that $\nu_2 = \nu_1[x := c]$.

Note that each $[\![\tau]\!]^{\mathbb{V}}_D$, being a set of ordered pairs of valuations, is a *binary relation on valuations*. When $\mathbb{V}$ is understood, we will feel free to omit the superscript in $[\![\tau]\!]^{\mathbb{V}}_D$.

Example 2.2. Consider a set of variables $\mathbb{V} = \{x, y, z\}$ and a schema $\mathcal{S}$ with two binary relation names B and T, both of input arity one. In the rest of the example, assume that $\mathbf{dom} \supseteq \{1, 2, 3, 4, 5\}$ and that we have an instance D of $\mathcal{S}$ that assigns the relation names to the following binary relations:

$$D(B) = \{(1, 2), (1, 3), (2, 3), (3, 5)\} \text{ and } D(T) = \{(1, 4), (3, 5)\}.$$

Intuitively, you could think of B and T as relations of source-destination pairs of stations that could be reached by bus (B) or train (T) respectively.

[1] Pronounced as "eff-lif".

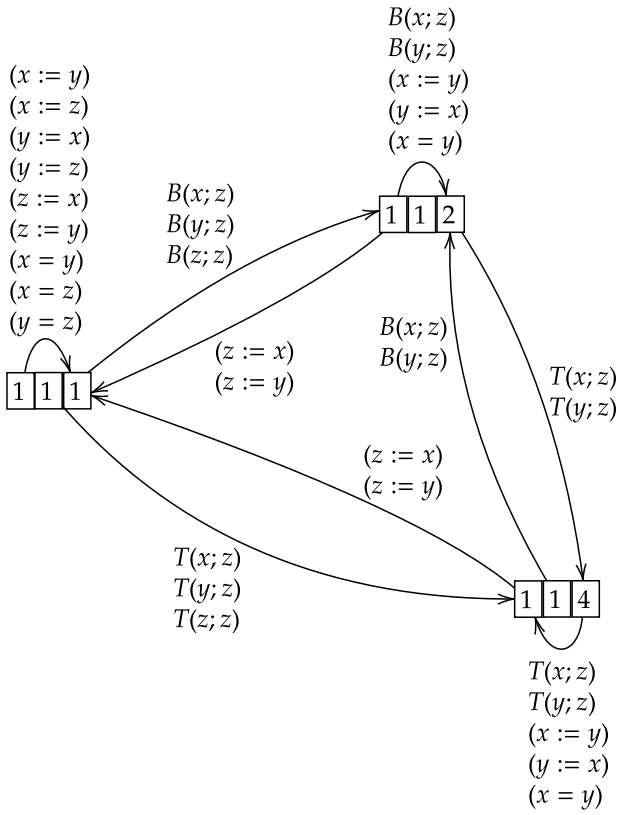

Figure 1: Part of graph view of the database considered in Example 2.2.

Figure 1 shows a tiny fragment of the graph view of D. It shows only three valuations and all labeled edges between these three valuations. We depict valuations by three consecutive squares with the first being the value of x, the second being the value of y, and the third being the value of z.

For another illustration of the same graph view, let us consider the following atomic expressions: $B(x;x)$, $B(x;y)$, $T(y;z)$, $(x := z)$, and $(x = z)$. Figure 2 depicts the entire binary relations on valuations $[\![\tau]\!]_D^{\mathbb{V}}$ for these five atomic expressions τ. For each of these examples, we give a table below that shows its semantics (i.e., pairs of $\mathbb{V}$-valuations). In this depiction, '$*$' means that the value of the variable could be anything in the domain, i.e., the variable in that valuation is not restricted to a specific value. Furthermore, when, in some pair, we put '$-$' in both slots for some variable u, we mean that the value for u could be anything on condition that it is the same on the left and right.

Syntax and semantics of FLIF: operators. The syntax of all FLIF expressions α (still over schema $\mathcal{S}$ and set of variables $\mathbb{V}$) is now given by the following grammar:

$$\alpha ::= \tau \mid \alpha\,;\alpha \mid \alpha \cup \alpha \mid \alpha - \alpha$$

$[\![B(x;x)]\!]^{\mathbb{V}}_D$

ν_1			ν_2		
x	y	z	x	y	z
1	–	–	2	–	–
1	–	–	3	–	–
2	–	–	3	–	–
3	–	–	5	–	–

$[\![B(x;y)]\!]^{\mathbb{V}}_D$

ν_1			ν_2		
x	y	z	x	y	z
1	$*$	–	1	2	–
1	$*$	–	1	3	–
2	$*$	–	2	3	–
3	$*$	–	3	5	–

$[\![T(y;z)]\!]^{\mathbb{V}}_D$

ν_1			ν_2		
x	y	z	x	y	z
–	1	$*$	–	1	4
–	3	$*$	–	3	5

$[\![(x := z)]\!]^{\mathbb{V}}_D$

ν_1			ν_2		
x	y	z	x	y	z
$*$	–	1	1	–	1
$*$	–	2	2	–	2
$*$	–	3	3	–	3
$*$	–	4	4	–	4
$*$	–	5	5	–	5

$[\![(x = z)]\!]^{\mathbb{V}}_D$

ν_1			ν_2		
x	y	z	x	y	z
1	–	1	1	–	1
2	–	2	2	–	2
3	–	3	3	–	3
4	–	4	4	–	4
5	–	5	5	–	5

Figure 2: Table view for the expressions considered in Example 2.2.

Here, τ ranges over atomic expressions over $\mathcal{S}$ and $\mathbb{V}$, as defined above. The semantics of the composition operator ';' is defined as follows:

$$[\![\alpha_1 \,;\, \alpha_2]\!]^{\mathbb{V}}_D = \{(\nu_1, \nu_2) \mid \exists \nu : (\nu_1, \nu) \in [\![\alpha_1]\!]^{\mathbb{V}}_D \text{ and } (\nu, \nu_2) \in [\![\alpha_2]\!]^{\mathbb{V}}_D\}$$

Note that we are simply taking the standard composition of two binary relations on valuations. Similarly, the semantics of the set operations are standard union and set difference on binary relations on valuations.

Example 2.3. Continuing Example 2.2, consider the expression $B(x;y)\,;T(y;x)$. Intuitively, this expression takes x as input and retrieves the possible values for x and y such that

(1) you can go from station x to station y by a bus, and moreover,
(2) you can go from station y to *a possibly different* station x by a train.

The next table of pairs of valuations shows the semantics of that FLIF expression, i.e., $[\![B(x;y)\,;T(y;x)]\!]^{\mathbb{V}}_D$.

ν_1			ν_2		
x	y	z	x	y	z
1	$*$	$-$	5	3	$-$
2	$*$	$-$	5	3	$-$

We see that FLIF expressions describe paths in the graph, in the form of source–target pairs. Composition is used to navigate through the graph, and to conjoin paths. Paths can be branched using union, and excluded using set difference.

Remark 2.4. Sometimes, in writing FLIF expressions, we omit parentheses around (sub)expressions involving composition since it is an associative operator. Also, we give precedence to composition over the set operations.

Example 2.5. Consider a simple Facebook abstraction with a single binary relation F of input arity one. When given a person as input, F returns all their friends. We assume that this relation is symmetric.

To illustrate the dynamic nature of FLIF, over just a single variable $\mathbb{V} = \{x\}$, the expression $F(x;x);F(x;x);F(x;x)$ describes all pairs (ν_1,ν_2) such that there is a path of length three from $\nu_1(x)$ to $\nu_2(x)$.

For another example, suppose, for an input person x (say, a famous person), we want to find all people who are friends with at least two friends of x. Formally, we want to navigate from a valuation ν_1 giving a value for x, to all valuations ν_2 giving values to variables y_1, y_2, and z, such that

- $\nu_2(y_1)$ and $\nu_2(y_2)$ are both friends with $\nu_1(x)$;
- $\nu_2(z)$ is friends with both $\nu_2(y_1)$ and $\nu_2(y_2)$; and
- $\nu_2(y_1) \neq \nu_2(y_2)$.

This can be done by the FLIF expression $\alpha - (\alpha\,;(y_1 = y_2))$, where α is the expression

$$F(x;y_1)\,;F(x;y_2)\,;F(y_1;z)\,;F(y_2;z_1)\,;(z = z_1).$$

Note that using the extra variable z_1 is needed, since using $F(y_2;z)$ instead would result in overwriting the value of the variable z set by the variable y_1.

Without the use of the extra variable, we could alternatively define α by the intersection $\alpha_1 \cap \alpha_2$, where α_i is the expression $F(x;y_1)\,;F(x;y_2)\,;F(y_i;z)$. We are using the intersection operator here, which is formally not part of FLIF as defined, but easily expressible as $\alpha_1 - (\alpha_1 - \alpha_2)$.

Remark 2.6. In the above example, it would be more efficient to simply write $\alpha\,;(y_1 \neq y_2)$. For simplicity, we have not added nonequality tests in FLIF as they are formally redundant in the presence of set difference, but they can easily be added in practice. The purpose of this paper is to introduce the formal foundation of FLIF; clearly, a practical language based on FLIF will include arithmetic comparisons and operations. □

The evaluation problem for FLIF expressions. Given that FLIF expressions navigate paths in the graph view of a database instance D, the natural use of FLIF is to provide an input valuation ν_{in} to an expression α, and ask for all output valuations ν_{out} such that $(\nu_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$. Formally, we define:

Definition 2.7 (The evaluation problem $Eval^{\mathbb{V}}_{\alpha}(D, \nu_{\text{in}})$ for FLIF expression α over $\mathbb{V}$)**.** Given a database instance D and a $\mathbb{V}$-valuation ν_{in}, the task is to compute the set

$$Eval^{\mathbb{V}}_{\alpha}(D, \nu_{\text{in}}) = \{\nu_{\text{out}} \mid (\nu_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D\}.$$

Assume we have effective access to the relations R of D, in the following sense. If R has input arity i and output arity o, and given an i-tuple t_1, we can effectively retrieve the set of o-tuples t_2 such that $t_1 \cdot t_2$ belongs to $D(R)$. Moreover, this set is assumed to be finite. Assuming such effective access, which is needed for the evaluation of atomic expressions of the form $R(\bar{x}; \bar{y})$, it is now obvious how more complex expressions can be evaluated. Indeed, other atomic expressions are just assignments or tests, and operations of FLIF are standard operations on binary relations. In Section 7, we will give an explicit description of this evaluation algorithm, for the "io-disjoint" fragment of FLIF, in terms of relational algebra plans. Nevertheless, the obvious evaluation algorithm described informally above can always be applied, also for FLIF expressions that are not io-disjoint.

Example 2.8. Recall the expression $F(x; x)\ ;\ F(x; x)\ ;\ F(x; x)$ from Example 2.5 over $\mathbb{V} = \{x\}$. On input a valuation ν_{in} on $\{x\}$, the evaluation will return all valuations ν_{out} on $\{x\}$ such that there is a path of length three from $\nu_{\text{in}}(x)$ to $\nu_{\text{out}}(x)$.

Next recall the expression $F(x; y_1)\ ;\ F(x; y_2)\ ;\ F(y_1; z)\ ;\ F(y_2; z_1)\ ;\ (z = z_1)$. On input valuation ν_{in} on $\mathbb{V} = \{x, y_1, y_2, z, z_1\}$, the evaluation will return all $\mathbb{V}$-valuations ν_{out} such that the tuples $(\nu_{\text{in}}(x), \nu_{\text{out}}(y_1))$, $(\nu_{\text{in}}(x), \nu_{\text{out}}(y_2))$, $(\nu_{\text{out}}(y_1), \nu_{\text{out}}(z))$, $(\nu_{\text{out}}(y_2), \nu_{\text{out}}(z))$ belong to relation F, and moreover $\nu_{\text{out}}(z_1) = \nu_{\text{out}}(z)$ and $\nu_{\text{out}}(x) = \nu_{\text{in}}(x)$. Note in particular that the values provided by ν_{in} for y_1, y_2, z, and z_1 are irrelevant; only the input value $\nu_{\text{in}}(x)$ counts. Similarly, recall the expression

$$F(x; y_1)\ ;\ F(x; y_2)\ ;\ F(y_1; z) \cap F(x; y_1)\ ;\ F(x; y_2)\ ;\ F(y_2; z)$$

over $\mathbb{V} = \{x, y_1, y_2, z\}$. On input a $\mathbb{V}$-valuation ν_{in}, the evaluation will return all $\mathbb{V}$-valuations ν_{out} such that the tuples $(\nu_{\text{in}}(x), \nu_{\text{out}}(y_1))$, $(\nu_{\text{in}}(x), \nu_{\text{out}}(y_2))$, $(\nu_{\text{out}}(y_1), \nu_{\text{out}}(z))$, $(\nu_{\text{out}}(y_2), \nu_{\text{out}}(z))$ belong to the relation F, and moreover $\nu_{\text{out}}(x) = \nu_{\text{in}}(x)$.

In contrast, consider the expression

$$F(x; y_1)\ ;\ F(y_1; z) \cap F(x; y_2)\ ;\ F(y_2; z).$$

Now on input a valuation ν_{in} on $\mathbb{V} = \{x, y_1, y_2, z\}$, the evaluation will return all $\mathbb{V}$-valuations ν_{out} such that the tuples $(\nu_{\text{in}}(x), \nu_{\text{in}}(y_1))$, $(\nu_{\text{in}}(x), \nu_{\text{in}}(y_2))$, $(\nu_{\text{in}}(y_1), \nu_{\text{out}}(z))$, $(\nu_{\text{in}}(y_2), \nu_{\text{out}}(z))$ belong to the relation F, and moreover, ν_{out} and ν_{in} agree on $\{x, y_1, y_2\}$. So for this expression, not just $\nu_{\text{in}}(x)$, but also $\nu_{\text{in}}(y_1)$ and $\nu_{\text{in}}(y_2)$ are important values for the evaluation problem. This behavior can be traced back to Definition 2.1, which requires $\nu_1(y_2) = \nu_2(y_2)$ for any pair $(\nu_1, \nu_2) \in [\![F(x; y_1)]\!]^{\mathbb{V}}_D$ as well as $[\![F(y_1; z)]\!]^{\mathbb{V}}_D$. Similarly, $\nu_1(y_1) = \nu_2(y_1)$ for any pair $(\nu_1, \nu_2) \in [\![F(x; y_2)]\!]^{\mathbb{V}}_D$ as well as $[\![F(y_2; z)]\!]^{\mathbb{V}}_D$.

3. Executable FO

Let us recall the language known as executable FO (cf. the Introduction). Executability of formulas is a syntactic notion. In the literature, a lot of work has focused on the problem of trying to rewrite arbitrary FO formulas into executable form [NL04, RSU95, Li03, DLN07, CM08b, BLT15, BtCT16, CMRU17]. However, in this paper, we are focusing instead on using executable FO as a gauge for accessing the expressiveness of our new language FLIF. (Indeed, we will show that FLIF and executable FO are equivalent.) Hence, in this paper, we work only with executable FO formulas and not with arbitrary FO formulas.

The notion of when a formula is executable is defined relative to a set of variables $\mathcal{V}$, which specifies the variables for which input values are already given. Beware (in line with established work in the area [Li03, NL04]) that the notion of executability here is syntactic, and dependent on how subformulas are ordered within the formula. One may think of the notion of executability discussed in this paper as a "left-to-right" executability, which shall be clear from the following examples. Indeed, we begin with a few examples.

Example 3.1.

- Let φ be the formula $R(x; y)$. As mentioned above, this notation makes clear that the input arity of R is one. If we provide an input value for x, then the database will give us all y values such that $R(x, y)$ holds. Indeed, φ will turn out to be $\{x\}$-executable. Giving a value for the first argument of R is mandatory, so φ is neither $\emptyset$-executable nor $\{y\}$-executable. However, it is certainly allowed to provide input values for both x and y; in that case we are merely testing if $R(x, y)$ holds for the given pair (x, y). Thus, φ is also $\{x, y\}$-executable. In general, a $\mathcal{V}$-executable formula will also be $\mathcal{V}'$-executable for any $\mathcal{V}' \supseteq \mathcal{V}$.
- Also, the formula $\exists y\, R(x; y)$ is $\{x\}$-executable. In contrast, the formula $\exists x\, R(x; y)$ is not, because even if a value for x is given as input, it will be ignored due to the existential quantification. In fact, the latter formula is not $\mathcal{V}$-executable for any $\mathcal{V}$.
- The formula $R(x; y) \wedge S(y; z)$ is $\{x\}$-executable, intuitively because each y returned by the formula $R(x; y)$ can be fed into the formula $S(y; z)$, which is $\{y\}$-executable in itself. In contrast, the *semantically equivalent* formula $S(y; z) \wedge R(x; y)$ is not $\{x\}$-executable, because we need a value for y to execute the formula $S(y; z)$. However, the entire formula is $\{y, x\}$-executable.
- The formula $R(x; y) \vee S(x; z)$ is not $\{x\}$-executable, because any y returned by $R(x; y)$ would already satisfy the formula, leaving the variable z unconstrained. This would lead to an infinite number of satisfying valuations. The formula is neither $\{x, z\}$-executable; if $S(x, z)$ holds for the given values for x and z, then y is left unconstrained. Of course, the formula is $\{x, y, z\}$-executable.
- For a similar reason, $\neg R(x; y)$ is only $\mathcal{V}$-executable for $\mathcal{V}$ containing x and y.

$\mathcal{V}$-executable Formulas. We now define, formally, for any set of variables $\mathcal{V}$, the set of $\mathcal{V}$-executable formulas are defined as follows. Our definition closely follows the original definition by Nash and Ludäscher [NL04]; we only add equalities and constants to the language.

- An equality $x = y$, for variables x and y, is $\mathcal{V}$-executable if at least one of x and y belongs to $\mathcal{V}$.
- An equality $x = c$, for a variable x and a constant c, is always $\mathcal{V}$-executable.

- A relation atom $R(\bar{x};\bar{y})$ is $\mathcal{V}$-executable if $X \subseteq \mathcal{V}$, where X is the set of variables from $\bar{x}$.
- A negation $\neg\varphi$ is $\mathcal{V}$-executable if φ is, and moreover $\mathit{fvars}(\varphi) \subseteq \mathcal{V}$.
- A conjunction $\varphi \wedge \psi$ is $\mathcal{V}$-executable if φ is, and moreover ψ is $\mathcal{V} \cup \mathit{fvars}(\varphi)$-executable.
- A disjunction $\varphi \vee \psi$ is $\mathcal{V}$-executable if both φ and ψ are, and moreover $\mathit{fvars}(\varphi) \triangle \mathit{fvars}(\psi) \subseteq \mathcal{V}$. Here, $\triangle$ denotes symmetric difference.
- An existential quantification $\exists x\, \varphi$ is $\mathcal{V}$-executable if φ is $\mathcal{V} - \{x\}$-executable.

Note that universal quantification is not part of the syntax of executable FO.

Example 3.2. Recall the query considered in Example 2.5, asking for all triples (y_1, y_2, z) such that, for some input x, we have $F(x; y_1)$, $F(x; y_2)$, $F(y_1; z)$, $F(y_2; z)$, and y_1 and y_2 are different. The natural FO formula for this query is indeed $\{x\}$-executable:

$$F(x; y_1) \wedge F(x; y_2) \wedge F(y_1; z) \wedge F(y_2; z) \wedge \neg(y_1 = y_2).$$

Note that the above executable FO formula and FLIF expression from Example 2.5 are quite similar in their structure. The main difference is the use of the extra variable z_1 which was explained in Example 2.5.

Remark 3.3. Continuing Remark 1.1, in an extended setting where multiple access patterns are possible for the same relation, the simple syntax we use both in FLIF and in executable FO needs to be changed. Instead of relation atoms of the form $R(\bar{x};\bar{y})$ we would use adornments, which is a standard syntax in the literature on access limitations. For example, if a ternary relation R can be accessed by giving inputs to the first two arguments, or to the first and the third, then both $R^{iio}(x, y, z)$ and $R^{ioi}(x, y, z)$ would be allowed relation atoms.

Given an FO formula φ and a finite set of variables $\mathcal{V}$ such that φ is $\mathcal{V}$-executable, we describe the following task:

Definition 3.4 (The evaluation problem $\mathit{Eval}_{\varphi,\mathcal{V}}(D, \nu_{\text{in}})$ for φ with input variables $\mathcal{V}$)**.** Given a database instance D and a valuation ν_{in} on $\mathcal{V}$, compute the set of all valuations ν on $\mathcal{V} \cup \mathit{fvars}(\varphi)$ such that $\nu_{\text{in}} \subseteq \nu$ and $D, \nu \models \varphi$.

As mentioned in the Introduction, this problem is known to be solvable by a relational algebra plan respecting the access patterns. In particular, if D is finite, the output is always finite: each valuation ν in the output can be shown to take only values in $\text{adom}(D) \cup \nu_{\text{in}}(\mathcal{V})$.[2]

3.1. **From Executable FO to FLIF.** After introducing FLIF and executable FO, we observe that executable FO formulas translate rather nicely to FLIF expression as given by the following Theorem.

Theorem 3.5. *Let φ be a $\mathcal{V}$-executable formula over a schema $\mathcal{S}$. There exists an FLIF expression α over $\mathcal{S}$ and a set of variables $\mathbb{V} \supseteq \mathit{fvars}(\varphi) \cup \mathcal{V}$ such that for every D, $\mathcal{V}$-valuation ν_{in}, and $\mathbb{V}$-valuation ν'_{in} with $\nu'_{\text{in}} \supseteq \nu_{\text{in}}$, we have*

$$\mathit{Eval}_{\varphi,\mathcal{V}}(D, \nu_{\text{in}}) = \{\nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} \mid (\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D\}.$$

Example 3.6. Before giving the proof, we give a few examples. Note that in all the following examples, we only consider sets of input variables $\mathcal{V}$ with $\mathcal{V} \subseteq \mathit{fvars}(\varphi)$.

[2]Actually, a stronger property can be shown: only values that are "accessible" from ν_{in} in D can be taken [BLtCT16], and if this accessible set is finite, the output of the evaluation problem is finite.

- Suppose φ is $R(x;y)$ with input variable x. Then, as expected, α can be taken to be $R(x;y)$. Suppose we have the same formula with $\mathcal{V} = \{x, y\}$. Intuitively, the formula asks for outputs (u) where u equals y. Hence, α can be taken to be $R(x;u);(u = y)$. Note that the FLIF expression $R(x;y)$ is not a correct translation since the value of y may change from the value given by $\mathcal{V}$.
- Now, consider $T(x;x,y)$, again with input variable x. Intuitively, the formula asks for outputs (u, y) where u equals x. Hence, a suitable FLIF translation is $T(x;u,y)\,;(u = x)$. Note that the FLIF expression $T(x;x,y)$ is not semantically equivalent since the value of x is changeable due to the dynamic semantics of FLIF.
- If φ is $R(x;y) \wedge S(y;z)$, still with input variable x, we can take $R(x;y)\,;S(y;z)$ for α. The same expression also serves for the formula $\exists y\,\varphi$.
- Suppose φ is $R(x;x) \vee S(y;)$ with $\mathcal{V} = \{x, y\}$. For $\mathcal{V} \cap \mathit{fvars}(R(x;x))$, we translate $R(x;x)$ to $R(x;u)\,;(x = u)$. Similarly, $S(y;)$ is translated to $S(y;)$. Then, the final α can be taken to be $R(x;u)\,;(x = u) \cup S(y;)$.
- A new trick must be used for negation. For example, if φ is $\neg R(x;y)$ with $\mathcal{V} = \{x, y\}$, then α can be taken to be $(u := 42) \;-\; R(x;u)\,;(u = y)\,;(u := 42)$. Composing each side of '$-$' with the same dummy assignment to u is required since the value of the u in the second operand should not affect the result of the needed negation.

Proof Sketch of Theorem 3.5. We only describe the translation; its correctness is proven in Section 6.1.

If φ is a relation atom $R(\bar{x};\bar{y})$, then α is $R(\bar{x};\bar{z});\xi$, where $\bar{z}$ is obtained from $\bar{y}$ by replacing each variable from $\mathcal{V}$ by a fresh variable. The expression ξ consists of the composition of all equalities $(y_i = z_i)$ where y_i is a variable from $\bar{y}$ that is in $\mathcal{V}$ and z_i is the corresponding fresh variable.

If φ is $x = y$, then α is $(x = y)$.

If φ is $x = c$, then α is $(x = c)$.

If φ is $\varphi_1 \wedge \varphi_2$, then by induction we have an expression α_1 for φ_1 and $\mathcal{V} \cap \mathit{fvars}(\varphi_1)$, and an expression α_2 for φ_2 and $(\mathcal{V} \cup \mathit{fvars}(\varphi_1)) \cap \mathit{fvars}(\varphi_2)$. Now α can be taken to be $\alpha_1\,;\alpha_2$.

If φ is $\exists x\,\varphi_1$, then without loss of generality we may assume that $x \notin \mathcal{V}$. By induction, we have an expression α_1 for φ_1 and $\mathcal{V}$. This expression also works for φ.

If φ is $\varphi_1 \vee \varphi_2$, then by induction we have an expression α_i for φ_i and $\mathcal{V}$, for $i = 1, 2$. Now α can be taken to be $\alpha_1 \cup \alpha_2$.

Finally, if φ is $\neg\varphi_1$, then by induction we have an expression α_1 for φ_1 and $\mathcal{V}$. Fix an arbitrary constant c, and let ξ be the composition of all expressions $(z := c)$ for $z \in \mathit{vars}(\alpha_1) - \mathcal{V}$. (If that set is empty, we add an extra fresh variable.) Then α can be taken to be $\xi \;-\; \alpha_1\,;\xi$. □

3.2. **From FLIF to executable FO.** The previous translation shows that FLIF is expressive enough, in the sense that executable FO formulas can be translated into FLIF expressions such that they evaluate to the same set of valuations starting from the same assignment. It turns out that the converse translation is also possible, so, FLIF exactly matches executable FO in expressive power.

Actually, two distinct translations from FLIF to executable FO are possible:

(1) A somewhat rough translation, which translates every FLIF expression on a set of variables $\mathbb{V}$ to an equivalent $\mathbb{V}$-executable formula that uses thrice the number of variables in $\mathbb{V}$;
(2) A much finer translation, which often results in $\mathcal{V}$-executable formulas with a much smaller set $\mathcal{V}$ than the entire $\mathbb{V}$. This set $\mathcal{V}$ will consist of the "input variables" of the given FLIF expression. We will this idea further in Sections 4 and 5.

Next, we proceed with the rough translation. Assume $\mathbb{V} = \{x_1, \ldots, x_n\}$. Since the semantics of FLIF expressions on $\mathbb{V}$ involves pairs of $\mathbb{V}$-valuations, we introduce a copy $\mathbb{V}_y = \{y_1, \ldots, y_n\}$ disjoint from $\mathbb{V}$. For clarity, we also write $\mathbb{V}_x$ for $\mathbb{V}$. By FO[k] we denote the fragment of first-order logic that uses only k distinct variables [Lib04].

The following proposition is a variant of a result shown in our companion paper on LIF [ABS+23, Proposition 7.9]. That result is for a larger language LIF, but it does not talk about executability.

Proposition 3.7. *Let $\mathcal{S}$ be a schema, and $\mathbb{V}_x$ a set of n variables. Then, for every FLIF expression α over $\mathcal{S}$ and $\mathbb{V}_x$, there exists a $\mathbb{V}_x$-executable* FO[$3n$] *formula φ_α over $\mathcal{S}$ with free variables in $\mathbb{V}_x \cup \mathbb{V}_y$ such that*

$$(\nu_1, \nu_2) \in [\![\alpha]\!]^{\mathbb{V}}_D \quad \leftrightarrow \quad D, (\nu_1 \cup \nu_2') \models \varphi_\alpha,$$

where ν_2' is the $\mathbb{V}_y$-valuation such that $\nu_2'(y_i) = \nu_2(x_i)$ for $i = 1, \ldots, n$.

Proof. The proof is by induction on the structure of α. First, we introduce a third copy $\mathbb{V}_z = \{z_1, \ldots, z_n\}$ of $\mathbb{V}$. Moreover, for every $u, v \in \{x, y, z\}$ we define ρ_{uv} as follows:

$$\rho_{uv} : \mathbb{V}_u \to \mathbb{V}_v : u_i \mapsto v_i$$

Using these functions, we can translate a valuation ν on $\mathbb{V} = \mathbb{V}_x$ to a corresponding valuation on $\mathbb{V}_u$ with $u \in \{y, z\}$. Clearly, the function composition in $\nu \circ \rho_{ux}$ does this job.

In the first part of the proof, we actually show a stronger statement by induction, namely that for each α and for every $u \neq v \in \{x, y, z\}$ there is a formula φ^{uv}_α in FO[$\mathbb{V}_x \cup \mathbb{V}_y \cup \mathbb{V}_z$] with set of free variables equal to $\mathbb{V}_u \cup \mathbb{V}_v$ such that for every D,

$$(\nu_1, \nu_2) \in [\![\alpha]\!]^{\mathbb{V}}_D \quad \leftrightarrow \quad D, (\nu_1 \circ \rho_{ux} \cup \nu_2 \circ \rho_{vx}) \models \varphi^{uv}_\alpha.$$

Since the notations x, y, z, u and v are taken, we use notations a, b and c for variables and d for constants.

- $\alpha = R(\overline{a}; \overline{b})$. Take φ^{uv}_α to be $R(\rho_{xu}(\overline{a}); \rho_{xv}(\overline{b})) \wedge \bigwedge_{c \notin \overline{b}} \rho_{xu}(c) = \rho_{xv}(c)$.
- $\alpha = (a = b)$. Take φ^{uv}_α to be $\rho_{xu}(a) = \rho_{xu}(b) \wedge \bigwedge_{c \in \mathbb{V}_x} \rho_{xu}(c) = \rho_{xv}(c)$.
- $\alpha = (a = d)$. Take φ^{uv}_α to be $\rho_{xu}(a) = d \wedge \bigwedge_{c \in \mathbb{V}_x} \rho_{xu}(c) = \rho_{xv}(c)$.
- $\alpha = (a := b)$. Take φ^{uv}_α to be $\rho_{xv}(a) = \rho_{xu}(b) \wedge \bigwedge_{c \in \mathbb{V}_x - \{a\}} \rho_{xu}(c) = \rho_{xv}(c)$.
- $\alpha = (a := d)$. Take φ^{uv}_α to be $\rho_{xv}(a) = d \wedge \bigwedge_{c \in \mathbb{V}_x - \{a\}} \rho_{xu}(c) = \rho_{xv}(c)$.
- $\alpha = \alpha_1 \cup \alpha_2$. Take φ^{uv}_α to be $\varphi^{uv}_{\alpha_1} \vee \varphi^{uv}_{\alpha_2}$.
- $\alpha = \alpha_1 - \alpha_2$. Take φ^{uv}_α to be $\varphi^{uv}_{\alpha_1} \wedge \neg \varphi^{uv}_{\alpha_2}$.
- $\alpha = \alpha_1 \,;\, \alpha_2$. Let $w \in \{x, y, z\} - \{u, v\}$. Take φ^{uv}_α to be $\exists w_1 \ldots \exists w_n \, (\varphi^{uw}_{\alpha_1} \wedge \varphi^{wv}_{\alpha_2})$.

In the rest of the proof, we verify that φ^{uv}_α is indeed $\mathbb{V}_u$-executable. As for the atomic FLIF expressions, this is clear.

In case $\alpha = \alpha_1 \cup \alpha_2$, we know by induction that both $\varphi^{uv}_{\alpha_1}$ and $\varphi^{uv}_{\alpha_2}$ are $\mathbb{V}_u$-executable. For φ^{uv}_α to be $\mathbb{V}_u$-executable, it must be the case that $\mathit{fvars}(\varphi^{uv}_{\alpha_1}) \triangle \mathit{fvars}(\varphi^{uv}_{\alpha_2}) \subseteq \mathbb{V}_u$ which is trivial since $\mathit{fvars}(\varphi^{uv}_{\alpha_1}) = \mathbb{V}_u \cup \mathbb{V}_v = \mathit{fvars}(\varphi^{uv}_{\alpha_2})$, so $\mathit{fvars}(\varphi^{uv}_{\alpha_1}) \triangle \mathit{fvars}(\varphi^{uv}_{\alpha_2}) = \emptyset$.

Now, consider the case $\alpha = \alpha_1 - \alpha_2$. We know by induction that both $\varphi^{uv}_{\alpha_1}$ and $\varphi^{uv}_{\alpha_2}$ are $\mathbb{V}_u$-executable. For φ^{uv}_{α} to be $\mathbb{V}_u$-executable, it must be the case that $\mathit{fvars}(\varphi^{uv}_{\alpha_2}) \subseteq \mathit{fvars}(\varphi^{uv}_{\alpha_1})$ which is true since $\mathit{fvars}(\varphi^{uv}_{\alpha_1}) = \mathbb{V}_u \cup \mathbb{V}_v = \mathit{fvars}(\varphi^{uv}_{\alpha_2})$.

Finally, consider the case $\alpha = \alpha_1 \,;\, \alpha_2$. We know by induction that $\varphi^{uw}_{\alpha_1}$ is $\mathbb{V}_u$-executable and $\varphi^{wv}_{\alpha_2}$ is $\mathbb{V}_w$-executable. It is clear that $\mathbb{V}_w \subseteq \mathit{fvars}(\varphi^{uw}_{\alpha_1})$, consequently, the formula $\varphi^{uw}_{\alpha_1} \wedge \varphi^{wv}_{\alpha_2}$ is $\mathbb{V}_u$-executable. Hence, since $\mathbb{V}_u$ and $\mathbb{V}_u$ are disjoint, the same formula is $(\mathbb{V}_u - \mathbb{V}_w)$-executable which is sufficient to show that φ^{uv}_{α} itself is $\mathbb{V}_u$-executable. □

Although the previous translations show that FLIF and executable FO are effectively equivalent in expressive power, the translation from FLIF to executable FO overlooks some of the interesting relations between both formalisms and moreover, it uses lots of variables unnecessarily. This is best shown by example.

Example 3.8. Consider the FLIF expression $\alpha = R(x_1; x_1) \,;\, R(x_1; x_1) \,;\, R(x_1; x_1)$ where $\mathbb{V}_x = \{x_1\}$. According to the procedure given in the proof of Proposition 3.7, the resultant φ_α would be

$$\exists z_1 \Big[R(x_1; z_1) \wedge \exists x_1 \big[R(z_1; x_1) \wedge R(x_1; y_1) \big] \Big].$$

In contrast, consider the FLIF expression $\alpha = R(x_1; x_2) \,;\, R(x_2; x_3) \,;\, R(x_3; x_4)$ where $\mathbb{V}_x = \{x_1, x_2, x_3, x_4\}$. Now φ_α would be

$$\begin{aligned}\exists z_1 \exists z_2 \exists z_3 \exists z_4 \Big[& R(x_1; z_2) \wedge (z_1 = x_1) \wedge (z_3 = x_3) \wedge (z_4 = x_4) \wedge \\ & \exists x_1 \exists x_2 \exists x_3 \exists x_4 \big[R(z_2; x_3) \wedge (x_1 = z_1) \wedge (x_2 = z_2) \wedge (x_4 = z_4) \wedge \\ & \qquad R(x_3; y_4) \wedge (y_1 = x_1) \wedge (y_2 = x_2) \wedge (y_3 = x_3) \big] \Big].\end{aligned}$$

However, it is clear that taking φ_α to be $R(x_1; x_2) \wedge R(x_2; x_3) \wedge R(x_3; x_4)$ would work fine, in the sense that given an arbitrary $\{x_1\}$-valuation ν and any $\mathbb{V}_x$-valuation ν' that is an extension of ν (i.e., $\nu' \supseteq \nu$), α and φ_α would evaluate to the same set of $\mathbb{V}_x$-valuations as stated below (where D below is an arbitrary instance):

$$\mathit{Eval}_{\varphi_\alpha, \{x_1\}}(D, \nu) = \mathit{Eval}^{\mathbb{V}_x}_{\alpha}(D, \nu')$$

This shows that the values provided for $\{x_2, x_3, x_4\}$ in ν' to evaluate the expression α are not important since their values would be overwritten regardless of what ν' sets them to. Stated differently, variables x_2, x_3, x_4 are *outputs* of α, but not *inputs*; the only input variables for α is x_1.

In the next section, we develop the notions of input and output variables of FLIF expressions more formally. Then in Section 5.2, we will give an improved translation from FLIF to executable FO taking inputs into account.

4. Inputs and outputs of forward LIF

In this section, we introduce inputs and outputs of FLIF expressions. In every expression, we can identify the *input* and the *output* variables. Intuitively, the output variables are those that can change value along the execution path; the input variables are those whose values at the beginning of the path are needed in order to know the possible values for the output variables. These intuitions will be formalized below. We first give some examples.

Table 1: Input and output variables of FLIF expressions. In the case of $R(\bar{x};\bar{y})$, the set X is the set of variables in $\bar{x}$, and the set Y is the set of variables in $\bar{y}$. Recall that $\triangle$ is symmetric difference.

α	$I(\alpha)$	$O(\alpha)$
$R(\bar{x};\bar{y})$	X	Y
$(x = y)$	$\{x, y\}$	$\emptyset$
$(x := y)$	$\{y\}$	$\{x\}$
$(x = c)$	$\{x\}$	$\emptyset$
$(x := c)$	$\emptyset$	$\{x\}$
$\alpha_1; \alpha_2$	$I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1))$	$O(\alpha_1) \cup O(\alpha_2)$
$\alpha_1 \cup \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1) \cup O(\alpha_2)$
$\alpha_1 - \alpha_2$	$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$	$O(\alpha_1)$

Example 4.1.

- In both expressions given for α from Example 2.5, the only input variable is x, and the other variables are output variables.
- FLIF, in general, allows expressions where a variable is both input and output. For example, consider the relation *Swap* of input arity two that holds of quadruples of the form (a, b, b, a) for a and b in the **dom**, so the values of the first two arguments are swapped in the second two. Then, using the expression $Swap(x, y; x, y)$ would result in having the values of x and y swapped. Formally, this expression defines all pairs of valuations (ν_1, ν_2) such that $\nu_2(x) = \nu_1(y)$ and $\nu_2(y) = \nu_1(x)$ (and ν_2 agrees with ν_1 on all other variables).
- On the other hand, for the expression $R(x; y_1) \cup S(x; y_2)$, the output variables are y_1 and y_2. Indeed, consider an input valuation ν_1 with $\nu_1(x) = a$. The expression pairs ν_1 either with a valuation giving a new value for y_1, or with a valuation giving a new value for y_2. However, y_1 and y_2 are also input variables (together with x). Indeed, when pairing ν_1 with a valuation ν_2 that sets y_2 to some b for which $S(a, b)$ holds, we must know the value of $\nu_1(y_1)$ so as to preserve it in ν_2. A similar argument holds for y_2. □

The semantic properties that we gave above as intuitions for the notions of inputs and outputs are undecidable in general (see related work Section 8). Here, we will work with syntactic approximations.

Definition 4.2. For any FLIF expression α, its sets $I(\alpha)$ and $O(\alpha)$ of input and output variables are defined in Table 1.

Note that previously we have used $vars(\alpha)$ to denote the set of all variables occurring in the expression α. Since FLIF has no explicit quantification, this is precisely the union of $I(\alpha)$ and $O(\alpha)$. From now on, we will also refer to this set as the *free variables*.

Next we establish three propositions that show that our definition of inputs and outputs, which is purely syntactic, reflects actual properties of the semantics. (See Section 8 on related work for their proofs.)

The first proposition confirms an intuitive property and can be straightforwardly verified by induction.

Proposition 4.3 (Inertia property)**.** *If $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ then ν_2 agrees with ν_1 outside $O(\alpha)$.*

The second proposition confirms, as announced earlier, that the semantics of expressions depends only on the free variables; outside $vars(\alpha)$, the binary relation $[\![\alpha]\!]_D$ is *cylindrical*, i.e., contains all possible data elements.[3] An illustration of this was already given in Figure 2, using the asterisk indications.

Proposition 4.4 (Free variable property)**.** *Let $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ and let ν_1' and ν_2' be valuations such that*

- *ν_1' agrees with ν_1 on $vars(\alpha)$, and*
- *ν_2' agrees with ν_2 on $vars(\alpha)$, and agrees with ν_1' outside $vars(\alpha)$.*

Then also $(\nu_1', \nu_2') \in [\![\alpha]\!]_D$.

The third proposition is the most important one. It confirms that the values for the input variables determine the values for the output variables.

Proposition 4.5 (Input-output determinacy)**.** *Let $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ and let ν_1' be a valuation that agrees with ν_1 on $I(\alpha)$. Then there exists a valuation ν_2' that agrees with ν_2 on $O(\alpha)$, such that $(\nu_1', \nu_2') \in [\![\alpha]\!]_D$.*

By inertia, we can see that the valuation ν_2' given by the above proposition is unique. Moreover, using the free variable property, we showed the input-output determinacy property is equivalent to the following alternative form.

Lemma 4.6 (Input-output determinacy, alternative form)**.** *Let $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ and let ν_1' be a valuation that agrees with ν_1 on $I(\alpha)$ as well as outside $O(\alpha)$. Then also $(\nu_1', \nu_2) \in [\![\alpha]\!]_D$.*

Intuitively, it is easier to work with the alternative form since we have to consider only three valuations instead of four in the original form.

Example 4.7. Let us denote the expression $R(x; y) \cup S(x; z)$ by α. The definitions in Table 1 yield that $O(\alpha) = \{y, z\}$ and $I(\alpha) = \{x, y, z\}$. Having y and z as input variables may at first sight seem counterintuitive. To see semantically why, say, y is an input variable for α, consider an instance D where S contains the pair $(1, 3)$. Consider the valuation $\nu_1 = \{(x, 1), (y, 2), (z, 0)\}$, and let $\nu_2 = \nu_1[z := 3]$. Clearly $(\nu_1, \nu_2) \in [\![S(x; z)]\!]_D \subseteq [\![\alpha]\!]_D$. However, if we change the value of y in ν_1, letting $\nu_1' = \nu_1[y := 4]$, then (ν_1', ν_2) neither belongs to $[\![S(x; z)]\!]_D$ nor to $[\![R(x; y)]\!]_D$ (due to inertia). Thus, input-output determinacy would be violated if y would not belong to $I(\alpha)$.

We are now in a position to formulate a new version of the FLIF evaluation problem that takes the inputs into consideration. Given an expression α, we consider the following task:[4]

Definition 4.8 (The evaluation problem $Eval_\alpha(D, \nu_{\text{in}})$ for α)**.** Given a database instance D and a valuation ν_{in} on $I(\alpha)$, the task is to compute the set

$$Eval_\alpha(D, \nu_{\text{in}}) = \{\nu_{\text{out}}|_{vars(\alpha)} \mid \exists \nu_{\text{in}}' : \nu_{\text{in}} \subseteq \nu_{\text{in}}' \text{ and } (\nu_{\text{in}}', \nu_{\text{out}}) \in [\![\alpha]\!]_D\}.$$

[3] This terminology is borrowed from cylindrical set algebra [IL84, VdB01].

[4] For a valuation ν on a set of variables X (possibly all variables), and a subset Y of X, we use $\nu|_Y$ to denote the restriction of ν to Y, i.e., $\nu|_Y$ is the function from the variables in Y to **dom** that agrees with ν on Y.

By inertia and input-output determinacy, the choice of ν'_{in} in the above definition of the output does not matter. We show this formally in the next Remark.

Remark 4.9. The above definition improves on Definition 2.7 in that it is formally independent of the encompassing universe $\mathbb{V}$ of variables; it intrinsically only depends on the input and output variables of α. Indeed, formally, given any FLIF expression α on $\mathbb{V}$, and any $\mathbb{V}$-valuation ν, it is not hard to see that the following equivalence holds:

$$Eval_\alpha(D, \nu|_{I(\alpha)}) = \{\nu'|_{vars(\alpha)} \mid \nu' \in Eval^{\mathbb{V}}_\alpha(D, \nu)\}$$

Proof. It suffices to show the '$\subseteq$' direction; the other direction is clear from the definitions. Let ν be a $\mathbb{V}$-valuation. Suppose that there exists an arbitrary $\mathbb{V}$-valuation ν_{in} such that $\nu_{\text{in}} \supseteq \nu|_{I(\alpha)}$ (i.e., $\nu_{\text{in}} = \nu$ on $I(\alpha)$) and $(\nu_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D$. We want to show that there exists a valuation ν'_{out} such that $(\nu, \nu'_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D$ and $\nu_{\text{out}} = \nu'_{\text{out}}$ on $vars(\alpha)$.

From the facts that $\nu = \nu_{\text{in}}$ on $I(\alpha)$ and that $(\nu_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D$, it follows by input-output determinacy that there exists a valuation ν'_{out} such that $(\nu, \nu'_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D$ and $\nu'_{\text{out}} = \nu_{\text{out}}$ on $O(\alpha)$. It remains to verify that $\nu'_{\text{out}} = \nu_{\text{out}}$ on $I(\alpha) - O(\alpha)$. Indeed, this is true since $\nu_{\text{out}} = \nu_{\text{in}} = \nu = \nu'_{\text{out}}$ on $I(\alpha) - O(\alpha)$, where the first and third equalities hold because of inertia and having both $(\nu_{\text{in}}, \nu_{\text{out}})$ and (ν, ν'_{out}) in $[\![\alpha]\!]^{\mathbb{V}}_D$. The middle equality follows from the fact that $\nu = \nu_{\text{in}}$ on $I(\alpha)$. □

Consider an FLIF expression α for which the set $O(\alpha)$ is disjoint from $I(\alpha)$. Then any pair $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$ satisfies that ν_1 and ν_2 are equal on $I(\alpha)$. Put differently, every $\nu_{\text{out}} \in Eval_\alpha(D, \nu_{\text{in}})$ is equal to ν_{in} on $I(\alpha)$; all that the evaluation does is expand the input valuation with output values for the new output variables. This makes the evaluation process for expressions α where $I(\beta) \cap O(\beta) = \emptyset$, for every subexpression β of α (including α itself), very transparent in which input slots remain intact while output slots are being filled. We call such expressions *io-disjoint*.

Example 4.10. Continuing Example 2.5 (friends), the expression $F(x; x)$ is obviously not io-disjoint. Evaluating this expression will overwrite the variable x with a friend of the person originally stored in x. In contrast, both expressions given for α in Example 2.5 are io-disjoint. Also the expression $R(x_1; x_2)\ ;\ R(x_2; x_3)\ ;\ R(x_3; x_4)$ from Example 3.8 is io-disjoint. Finally, the expression $R(x; y_1) \cup S(x; y2)$ already seen in Example 4.1 is not io-disjoint. □

Formally, we have the following useful property, which follows from inertia and input-output determinacy.

Proposition 4.11 (Identity property)**.** *Let α be an io-disjoint expression and let D be an instance. If $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, then also $(\nu_2, \nu_2) \in [\![\alpha]\!]_D$.*

Intuitively, the identity property holds because, if in ν_1 the output slots would accidentally already hold a correct combination of output values, then there will exist an evaluation of α that merely confirms these values. This property can be interpreted to say that io-disjoint expressions can be given a "static" semantics; we could say that a single valuation ν satisfies α when (ν, ν) belongs to the dynamic semantics. This brings io-disjoint expressions closer to the conventional static semantics (single valuations) of first-order logic. Indeed, this will be confirmed in the next Section.

Example 4.12. The identity property clearly need not hold for expressions that are not io-disjoint. For example, continuing the friends example, for the expression $F(x; x)$, a person need not be a friend of themselves.

The following proposition makes it easier to check if an expression is io-disjoint:

Proposition 4.13. *The following alternative definition of io-disjointness is equivalent to the definition given above:*

- *An atomic expression $R(\bar{x};\bar{y})$ is io-disjoint if $X \cap Y = \emptyset$, where X is the set of variables in $\bar{x}$, and Y is the set of variables in $\bar{y}$.*
- *Atomic expressions of the form $(x = y)$, $(x = c)$, $(x := y)$ or $(x := c)$ are io-disjoint.*
- *A composition $\alpha_1 \,;\, \alpha_2$ is io-disjoint if α_1 and α_2 are, and moreover $I(\alpha_1) \cap O(\alpha_2) = \emptyset$.*
- *A union $\alpha_1 \cup \alpha_2$ is io-disjoint if α_1 and α_2 are, and moreover $O(\alpha_1) = O(\alpha_2)$.*
- *A difference $\alpha_1 - \alpha_2$ is io-disjoint if α_1 and α_2 are, and moreover $O(\alpha_1) \subseteq O(\alpha_2)$.*

The fragment of io-disjoint expressions is denoted by FLIF^{io}. In the next section, we are going to show that FLIF^{io} is expressive enough, in the sense that FLIF expressions can be simulated by FLIF^{io} expressions that have the same set of input variables. Furthermore, we will give the improved translation from FLIF^{io} to executable FO, which takes inputs into account.

5. Io-disjoint FLIF

We begin this section by showing that any FLIF expression can be converted to an io-disjoint one. We will first discuss the problem and its complications by means of illustrative examples. After that, we formulate the precise theorem and give a constructive method to rewrite FLIF expressions into io-disjoint ones.

5.1. From FLIF to io-disjoint FLIF.

In this section, we are discussing a possible approach to translate general FLIF expressions into io-disjoint ones that simulate the original expressions; we also discuss what "simulate" can mean. For instance, we will see that we have to use extra variables in order to get io-disjointness. An appropriate notion of simulation will then involve renaming of output variables.

For example, we rewrite $R(x;x)$ to $R(x;y)$ and declare that the output value for x can now be found in slot y instead. This simple idea, however, is complicated when handling the different operators of FLIF. These complications are discussed next.

5.1.1. *Complications of Translation.*

When applying the simple renaming approach to the composition of two expressions, we must be careful, as an output of the first expression can be taken as input in the second expression. In that case, when renaming the output variable of the first expression, we must apply the renaming also to the second expression, but only on the input side. For example, $R(x;x) \,;\, S(x;x)$ is rewritten to $R(x;y) \,;\, S(y;z)$. Thus, the output x of the overall expression is renamed to z; the intermediate output x of the first expression is renamed to y, as is the input x of the second expression.

Obviously, we must also avoid variable clashes. For example, in $R(x;x) \,;\, S(y;y)$, when rewriting the subexpression $R(x;x)$, we should not use y to rename the output x to, as this variable is already in use in another subexpression.

Another subtlety arises in the rewriting of set operations. Consider, for example, the union $R(x;y) \cup S(x;z)$. As discussed in Example 4.10, this expression is not io-disjoint: the output variables are y and z, but these are also input variables, in addition to x. To make the expression io-disjoint, it does not suffice to simply rename y and z, say, to y_1 and z_1. We

can, however, add assignments to both sides in such a way to obtain a formally io-disjoint expression:

$$R(x; y_1)\,;(z_1 := z)\ \cup\ S(x; z_1)\,;(y_1 := y).$$

The above trick must also be applied to intermediate variables. For example, consider $T(;)\cup(S(;y)\,;R(y;y))$. Note that T is a nullary relation. This expression is not io-disjoint with y being an input variable as well as an output variable. The second term is readily rewritten to $S(;y_1);R(y_1;y_2)$ with y_2 the new output variable. Note that y_1 is an intermediate variable. The io-disjoint form becomes

$$T(;)\,;(y_1 := y)\,;(y_2 := y)\ \cup\ S(;y_1)\,;R(y_1;y_2).$$

In general, it is not obvious that one can always find a suitable variable to set intermediate variables from the other subexpression to. In our proof of the theorem we prove formally that this is always possible.

A final complication occurs in the treatment of difference. Intermediate variables used in the rewriting must be reset to the same value in both subexpressions, since the difference operator is sensitive to the values of all variables. For example, let α be the expression $S(;x)\,;R(x;u,x)\ -\ T(;)$. We have $I(\alpha)=O(\alpha)=\{x,u\}$. Suppose we want to rename the outputs x and u to x_1 and u_1 respectively. As before, the subexpression on the lhs of the difference operator is rewritten to $S(;x_2)\,;R(x_2;u_1,x_1)$ introducing an intermediate variable x_2. Also as before, x_1 and u_1 need to be added to the rewriting of $T(;)$ which does not have x and u as outputs. But the new complication is that x_2 needs to be reset to a common value (we use x here) for the difference of the rewritten subexpressions to have the desired semantics. We thus obtain the overall rewriting

$$S(;x_2)\,;R(x_2;u_1,x_1)\,;(x_2 := x)\ -\ T(;)\,;(x_2 := x)\,;(u_1 := u)\,;(x_1 := x).$$

5.1.2. *Statement of the theorem.* As the overall idea behind the above examples was to rename the output variables, our aim is clearly the following theorem, with ρ playing the role of the renaming:[5]

Theorem 5.1. *Let α be an FLIF expression and let ρ be a bijection from $O(\alpha)$ to a set of variables disjoint from $vars(\alpha)$. There exists an* FLIF$^{\text{io}}$ *expression β such that*

(1) $I(\beta)=I(\alpha)$*;*
(2) $O(\beta)\supseteq\rho(O(\alpha))$*; and*
(3) *for every instance D and every valuation ν_1, we have*

$$\{\nu_2|_{O(\alpha)}\mid(\nu_1,\nu_2)\in[\![\alpha]\!]^{\mathbb{V}}_D\}=\{\nu_2\circ\rho\mid(\nu_1,\nu_2)\in[\![\beta]\!]^{\mathbb{V}}_D\}.$$

Here, $\mathbb{V}$ is any set of variables containing $vars(\alpha)$ and $vars(\beta)$.

In the above theorem, we must allow $O(\beta)$ to be a superset of $\rho(O(\alpha))$ (rather than being equal to it), because we must allow the introduction of auxiliary (intermediate) variables. For example, let α be the expression $S(x;)-R(x;x)$. Note that $O(\alpha)$ is empty. Interpret S as holding bus stops and R as holding bus routes. Then α represents an information source with limited access pattern that takes as input x, and tests if x is a bus stop to where the bus would not return if we would take the bus at x. Assume, for the sake of contradiction, that there would exist an io-disjoint expression β as in the theorem, but

[5]We use $g\circ f$ for standard function composition ("g after f"). So, in the statement of the theorem, $\nu_2\circ\rho: O(\alpha)\to\mathbb{V}: x\mapsto\nu_2(\rho(x))$.

with $O(\beta) = O(\alpha) = \emptyset$. Since $I(\beta)$ must equal $I(\alpha) = \{x\}$, the only variable occurring in β is x. In particular, β can only mention R in atomic subexpressions of the form $R(x; x)$, which is not io-disjoint. We are forced to conclude that β cannot mention R at all. Such an expression, however, can never be a correct rewriting of α. Indeed, let D be an instance for which $[\![\alpha]\!]_D$ is nonempty. Hence $[\![\beta]\!]_D$ is nonempty as well. Now let D' be the instance with $D'(S) = D(S)$ but $D'(R) = \emptyset$. Then $[\![\alpha]\!]_{D'}$ becomes clearly empty, but $[\![\beta]\!]_{D'} = [\![\beta]\!]_D$ remains nonempty since β does not mention R.

5.1.3. *Variable renaming.* In the proof of our theorem we need a rigorous way of renaming variables in FLIF expressions. The following lemma allows us to do this. It confirms that expressions behave under variable renamings as expected. The proof by structural induction is straightforward.

As to the notation used in the lemma, recall that $\mathbb{V}$ is defined to be the universe of variables. For a permutation θ of $\mathbb{V}$, and an expression α, we use $\theta(\alpha)$ for the expression obtained from α by replacing every occurrence of any variable x by $\theta(x)$.

Lemma 5.2 (Renaming Lemma)**.** *Let α be an FLIF expression and let θ be a permutation of $\mathbb{V}$. Then for every instance D, we have*

$$(\nu_1, \nu_2) \in [\![\alpha]\!]_D \iff (\nu_1 \circ \theta, \nu_2 \circ \theta) \in [\![\theta(\alpha)]\!]_D.$$

5.1.4. *Rewriting procedure.* In order to be able to give a constructive proof of Theorem 5.1 by structural induction, a stronger induction hypothesis is needed. Specifically, to avoid clashes, we introduce a set W of forbidden variables. So we will actually prove the following statement:

Lemma 5.3. *Let α be an FLIF expression, let W be a set of variables, and let ρ be a bijection from $O(\alpha)$ to a set of variables disjoint from vars(α). There exists an* FLIF$^{\text{io}}$ *expression β such that*

(1) *$I(\beta) = I(\alpha)$;*
(2) *$O(\beta) \supseteq \rho(O(\alpha))$ and $O(\beta) - \rho(O(\alpha))$ is disjoint from W;*
(3) *for every instance D and every valuation ν_1, we have*

$$\{\nu_2|_{O(\alpha)} \mid (\nu_1, \nu_2) \in [\![\alpha]\!]_D\} = \{\nu_2 \circ \rho \mid (\nu_1, \nu_2) \in [\![\beta]\!]_D\}.$$

We proceed to formally describe an inductive rewriting procedure to produce β from α as prescribed by the above lemma. The procedure formalizes and generalizes the situations encountered in the examples discussed in the previous section. The correctness of the method is proven in Section 6.2.

Terminology. A bijection from a set of variables X to another set of variables is henceforth called a *renaming of X*.

Relation atom. If α is of the form $R(\bar{x}; \bar{y})$, then β equals $R(\bar{x}; \rho(\bar{y}))$.

Variable assignment. If α is of the form $x := t$, then β equals $\rho(x) := t$.

Equality test. If α is an equality test, we can take β equal to α.

Nullary expressions. An expression α is called *nullary* if it contains no variables, i.e., $vars(\alpha)$ is empty. Trivially, for nullary α, the desired β can be taken to be α itself. We will consider this to be an extra base case for the induction.

Composition. If α is of the form $\alpha_1 \,;\, \alpha_2$ then β equals $\beta_1 \,;\, \theta(\beta_2)$, where the constituents are defined as follows.

- Let $W_2 = W \cup vars(\alpha) \cup \rho(O(\alpha_1))$, and let ρ_2 be the restriction of ρ to $O(\alpha_2)$. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- Let $W_1 = W \cup vars(\alpha)$, and let ρ_1 be a renaming of $O(\alpha_1)$ such that
  - on $O(\alpha_1) \cap O(\alpha_2) \cap I(\alpha_2)$, the image of ρ_1 is disjoint from $vars(\alpha) \cup O(\beta_2)$ as well as from the image of ρ;
  - elsewhere, ρ_1 agrees with ρ.

  By induction, there exists an io-disjoint renaming of α_1 for W_1 and ρ_1; this yields β_1.
- θ is the permutation of $\mathbb{V}$ defined as follows. For every $y \in I(\alpha_2) \cap O(\alpha_1)$, we have

$$\theta(y) = \rho_1(y) \text{ and } \theta(\theta(y)) = y.$$

  Elsewhere, θ is the identity.

Union. If α is of the form $\alpha_1 \cup \alpha_2$ then β equals $(\beta_1 \,;\, \gamma_1 \,;\, \eta_1) \cup (\beta_2 \,;\, \gamma_2 \,;\, \eta_2)$ where the constituent expressions are defined as follows.

- Let $W_1 = W \cup vars(\alpha) \cup \rho(O(\alpha_2))$ and let ρ_1 be the restriction of ρ on $O(\alpha_1)$. By induction, there exists an io-disjoint rewriting of α_1 for W_1 and ρ_1; this yields β_1.
- Let $W_2 = W \cup vars(\alpha) \cup O(\beta_1)$ and let ρ_2 be the restriction of ρ on $O(\alpha_2)$. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- γ_1 is the composition of all $(\rho(y) := y)$ for $y \in O(\alpha_2) - O(\alpha_1)$, and γ_2 is defined symmetrically.
- If $O(\beta_2) - \rho_2(O(\alpha_2))$ (the set of "intermediate" variables in β_2) is empty, η_1 can be dropped from the expression. Otherwise, η_1 is the composition of all $(y := z)$ for $y \in O(\beta_2) - \rho(O(\alpha_2))$, with z a fixed variable chosen as follows.
  (a) If $O(\beta_1)$ is nonempty, take z arbitrarily from there.
  (b) Otherwise, take z arbitrarily from $vars(\alpha_2)$. We know $vars(\alpha_2)$ is nonempty, since otherwise α_2 would be nullary, so β_2 would equal α_2, and then $O(\beta_2)$ would be empty as well (extra base case), which is not the case.
- η_2 is defined symmetrically.

Difference. If α is of the form $\alpha_1 - \alpha_2$ then β equals $(\beta_1 \,;\, \gamma_1 \,;\, \eta_1 \,;\, \eta_2) - (\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2)$ where the constituent expressions are defined as follows.

- Let $W_1 = W \cup vars(\alpha)$ and let $\rho_1 = \rho$. By induction, there exists an io-disjoint rewriting of α_1 for W_1 and ρ; this yields β_1.
- Let $W_2 = W_1 \cup O(\beta_1)$ and let ρ_2 be a renaming of $O(\alpha_2)$ that agrees with ρ on $O(\alpha_1) \cap O(\alpha_2)$, such that the image of $\rho_2 - \rho_1$ is disjoint from W_2. By induction, there exists an io-disjoint rewriting of α_2 for W_2 and ρ_2; this yields β_2.
- γ_1 is the composition of all $(\rho_2(y) := y)$ for $y \in O(\alpha_2) - O(\alpha_1)$, and γ_2 is defined symmetrically.
- If $O(\beta_2) - \rho_2(O(\alpha_2))$ is empty, η_1 can be dropped from the expression. Otherwise, η_1 is the composition of all $(y := z)$ for $y \in O(\beta_2) - \rho_2(O(\alpha_2))$, with z a fixed variable chosen as follows.

(a) If $O(\alpha_1) \cap O(\alpha_2)$ is nonempty, take z arbitrarily from $\rho(O(\alpha_1) \cap O(\alpha_2))$.
(b) Otherwise, take z arbitrarily from $vars(\alpha_2)$ (which is nonempty by the same reasoning as given for the union case).

- η_2 is defined symmetrically.

5.1.5. *Necessity of variable assignment.* Our rewriting procedure intensively uses variable assignment. Is this really necessary? More precisely, suppose α itself does not use variable assignment. Can we still always find an io-disjoint rewriting β such that β does not use variable assignment either? Below, we answer this question negatively; in other words, the ability to do variable assignment is crucial for io-disjoint rewriting.

For our counterexample we work over the schema consisting of a nullary relation name S and a binary relation name T of input arity one. Let α be the expression $S(;) \cup T(x;x)$ and let ρ rename x to x_1. Note that our rewriting procedure would produce the rewriting

$$S(;)\,;(x_1 := x) \;\cup\; T(x;x_1),$$

indeed using a variable assignment $(x_1 := x)$ to ensure an io-disjoint expression.

For the sake of contradiction, assume there exists an expression β according to Theorem 5.1 that does not use variable assignment. Fix D to the instance where S is nonempty but T is empty. Then $[\![\alpha]\!]_D$ consists of all identical pairs of valuations. Take any valuation ν with $\nu(x) \neq \nu(x_1)$. Since $(\nu,\nu) \in [\![\alpha]\!]_D$, there should exist a valuation ν' with $\nu'(x_1) = \nu(x)$ such that $(\nu,\nu') \in [\![\beta]\!]_D$. Note that $\nu' \neq \nu$, since $\nu(x_1) \neq \nu(x)$. However, this contradicts the following two observations. Both observations are readily verified by induction. (Recall that D is fixed as defined above.)

(1) For every expression β without variable assignments, either $[\![\beta]\!]_D$ is empty, or $[\![\beta]\!]_D = [\![\gamma]\!]_D$ for some expression γ that does not mention T and that has no variable assignments.
(2) For every expression γ that does not mention T and that has no variable assignments, and any $(\nu_1,\nu_2) \in [\![\gamma]\!]_D$, we have $\nu_1 = \nu_2$.

5.2. **Improved translation from io-disjoint FLIF to Executable FO.** We now turn to the translation from FLIF^{io} to executable FO. Here, a rather straightforward equivalence is possible, since executable FO has an explicit quantification operation which is lacking in FLIF. Recall the evaluation problem for executable FO (Definition 3.4, and the evaluation problem for α (Definition 4.8).

Theorem 5.4. *Let α be an FLIF^{io} expression over a schema $\mathcal{S}$. There exists an $I(\alpha)$-executable FO formula φ_α over $\mathcal{S}$, with $fvars(\varphi_\alpha) = vars(\alpha)$, such that for every D and ν_{in}, we have $Eval_\alpha(D,\nu_{\text{in}}) = Eval_{\varphi_\alpha,I(\alpha)}(D,\nu_{\text{in}})$. The length of φ_α is linear in the length of α.*

Example 5.5. To illustrate the proof, consider the FLIF^{io} expression $R(x;y,u)\,;S(x;z,u)$. Procedurally, to evaluate the first expression, we retrieve values for the variables y and u that match the value given for the variable x in the relation R. We proceed to retrieve a (z,u)-binding from S for the given x, effectively overwriting the previous binding for u. Thus, a correct translation into executable FO is $(\exists u\, R(x;y,u)) \wedge S(x;z,u)$.

Interpreting relations as functions, this example can be likened to the following piece of code in Python:

```
y,u = R(x) ; z,u = S(x)
```

Table 2: Translation showing how FLIF^{io} embeds in executable FO. In the table, φ_i abbreviates φ_{α_i} for $i = 1, 2$.

α	φ_α
$R(\bar{x}; \bar{y})$	$R(\bar{x}; \bar{y})$
$(x = y)$	$x = y$
$(x := y)$	$x = y$
$x = c$	$x = c$
$x := c$	$x = c$
$\alpha_1; \alpha_2$	$(\exists x_1 \dots \exists x_k \, \varphi_1) \wedge \varphi_2$ where $\{x_1, \dots, x_k\} = O(\alpha_1) \cap O(\alpha_2)$
$\alpha_1 \cup \alpha_2$	$\varphi_1 \vee \varphi_2$
$\alpha_1 - \alpha_2$	$\varphi_1 \wedge \neg\varphi_2$

Indeed, formalisms such as FLIF, as well as its mother framework LIF [Ter19], dynamic logic [HKT00], and dynamic predicate logic [GS91] provide logical foundations for such programming constructs (and even natural language constructs).

For another example, consider the assignment $(x := y)$. This translates to $x = y$ considered as a $\{y\}$-executable formula. The equality test $(x = y)$ also translates to $x = y$, but considered as an $\{x, y\}$-executable formula.

Proof Sketch of Theorem 5.4. Table 2 shows the translation, which is almost an isomorphic embedding, except for the case of composition. The correctness of the translation for composition again hinges on inertia and input-output determinacy. The formal correctness proof, including the verification that φ_α is indeed $I(\alpha)$-executable, is given in Section 6.3. □

6. Correctness Proofs of Translation Theorems

6.1. **From Executable FO to FLIF.** In this section we prove Theorem 3.5, which is reformulated below for convenience.

Theorem 3.5. *Let φ be a $\mathcal{V}$-executable formula over a schema $\mathcal{S}$. There exists an FLIF expression α over $\mathcal{S}$ and a set of variables $\mathbb{V} \supseteq fvars(\varphi) \cup \mathcal{V}$ such that for every D, valuation ν_{in} on $\mathcal{V}$, and valuation ν'_{in} on $\mathbb{V}$ with $\nu'_{\text{in}} \supseteq \nu_{\text{in}}$, we have*

$$\{\nu|_{fvars(\varphi)\cup\mathcal{V}} \mid \nu_{\text{in}} \subseteq \nu \text{ and } D, \nu \models \varphi\} = \{\nu_{\text{out}}|_{fvars(\varphi)\cup\mathcal{V}} \mid (\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]^{\mathbb{V}}_D\}.$$

Proof. By structural induction. The containment from left to right is referred to as *completeness*, and the containment from right to left as *soundness*.

In the proof, we will omit the explicit definition of the set $\mathbb{V}$ and we take it to be the set of all variables mentioned in the constructed expression α. It is also worth noting that it follows from the statement of the theorem that α cannot change the values of the variables in $\mathcal{V}$. Precisely, for every ν_1, ν_2 such that $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, it must be the case that ν_1 agrees with ν_2 on $\mathcal{V}$.

Atoms. If φ is a relation atom $R(\bar{x};\bar{y})$, then α is $R(\bar{x};\bar{z})\,;\xi$, where $\bar{z}$ is obtained from $\bar{y}$ by replacing each variable from $\mathcal{V}$ by a fresh variable. The expression ξ consists of the composition of all equalities $(y_i = z_i)$ where y_i is a variable from $\bar{y}$ that is in $\mathcal{V}$ and z_i is the corresponding fresh variable. In what follows, let X, Y, and Z be the variables in $\bar{x}$, $\bar{y}$, and $\bar{z}$; respectively. Moreover, take ν_{in} to be an arbitrary valuation on $\mathcal{V}$, and ν'_{in} to be any valuation such that $\nu'_{\text{in}} \supseteq \nu_{\text{in}}$.

We first prove completeness. Let ν to be a valuation on $Y \cup \mathcal{V}$ such that $\nu_{\text{in}} \subseteq \nu$. Now, suppose that $D, \nu \models \varphi$. We want to verify that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$ and $\nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} = \nu$, which is clear when taking ν_{out} to be the valuation that agrees with ν_{in} on $\mathcal{V}$, agrees with ν on $\mathit{fvars}(\varphi) - \mathcal{V}$, agrees with ν'_{in} outside $\mathit{vars}(\alpha)$, and satisfies $\nu_{\text{out}}(z) = \nu(y)$ for every $y \in (Y \cap \mathcal{V})$ and its corresponding $z \in Z$.

To show soundness, suppose that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$. We want to verify that $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} \models \varphi$, which is clear given the semantics of α.

The cases where φ is of the form $x = y$ or $(x = c)$ are handled as already shown in the previous section; correctness is clear.

Conjunction. If φ is $\varphi_1 \wedge \varphi_2$, then by induction we have an expression α_1 for φ_1 and $\mathcal{V}$, and an expression α_2 for φ_2 and $\mathcal{V} \cup \mathit{fvars}(\varphi_1)$ (since φ_2 is $\mathcal{V} \cup \mathit{fvars}(\varphi_1)$-executable). We show that α can be taken to be $\alpha_1\,;\alpha_2$. Take ν_{in} to be an arbitrary valuation on $\mathcal{V}$, and ν'_{in} to be any valuation such that $\nu'_{\text{in}} \supseteq \nu_{\text{in}}$.

We first prove completeness. Let ν be a valuation on $\mathit{fvars}(\varphi) \cup \mathcal{V}$ such that $\nu_{\text{in}} \subseteq \nu$. Now, suppose that $D, \nu \models \varphi$. We want to verify that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$ and $\nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} = \nu$. Clearly, $D, \nu \models \varphi_1$ and $D, \nu \models \varphi_2$. By induction, there exists ν_1 such that $(\nu'_{\text{in}}, \nu_1) \in [\![\alpha_1]\!]_D$ and $\nu_1 = \nu$ on $\mathit{fvars}(\varphi_1) \cup \mathcal{V}$. From the last equality and also from induction, there exists ν_{out} such that $(\nu_1, \nu_{\text{out}}) \in [\![\alpha_2]\!]_D$ and $\nu_{\text{out}} = \nu_1 = \nu$ on $\mathit{fvars}(\varphi_2) \cup \mathcal{V} \cup \mathit{fvars}(\varphi_1) = \mathit{fvars}(\varphi) \cup \mathcal{V}$.

We next show soundness. Suppose that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$. We want to verify that $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} \models \varphi$, and that $\nu_{\text{out}} \supseteq \nu_{\text{in}}$. Clearly, there exists a valuation ν such that $(\nu'_{\text{in}}, \nu) \in [\![\alpha_1]\!]_D$ and $(\nu, \nu_{\text{out}}) \in [\![\alpha_2]\!]_D$. By induction, $D, \nu|_{\mathit{fvars}(\varphi_1)\cup\mathcal{V}} \models \varphi_1$ and $\nu \supseteq \nu_{\text{in}}$. Also by induction, $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi_2)\cup\mathit{fvars}(\varphi_1)\cup\mathcal{V}} \models \varphi_2$ and $\nu_{\text{out}} \supseteq \nu|_{\mathit{fvars}(\varphi_1)\cup\mathcal{V}}$. From the latter, we obtain $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} \models \varphi$. Showing that $\nu_{\text{out}} \supseteq \nu_{\text{in}}$ is clear.

Disjunction. If φ is $\varphi_1 \vee \varphi_2$, then by induction we have an expression α_i for φ_i and $(\mathit{fvars}(\varphi_1) \triangle \mathit{fvars}(\varphi_2)) \cup \mathcal{V}$ for $i = 1, 2$ (since $\mathit{fvars}(\varphi_1) \triangle \mathit{fvars}(\varphi_2) \subseteq \mathcal{V}$). We show that α can be taken to be $\alpha_1 \cup \alpha_2$. Take ν_{in} to be an arbitrary valuation on $\mathcal{V}$, and ν'_{in} to be any valuation such that $\nu'_{\text{in}} \supseteq \nu_{\text{in}}$.

We first prove completeness. Let ν be a valuation on $\mathit{fvars}(\varphi) \cup \mathcal{V}$ such that $\nu_{\text{in}} \subseteq \nu$. Now, suppose that $D, \nu \models \varphi$. We want to verify that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$ and $\nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} = \nu$. We only consider the case when $D, \nu \models \varphi_1$; the other is symmetric. By induction, there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha_1]\!]_D$ and $\nu_{\text{out}}|_{\mathit{fvars}(\varphi_1)\cup\mathcal{V}} = \nu$. Clearly, $\mathit{fvars}(\varphi_1) \cup \mathcal{V} = \mathit{fvars}(\varphi) \cup \mathcal{V}$ from the conditions on $\mathcal{V}$, and we are done.

To show soundness, suppose that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$. We want to verify that $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi)\cup\mathcal{V}} \models \varphi$, and that $\nu_{\text{out}} \supseteq \nu_{\text{in}}$. Again, we only consider the case when $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha_1]\!]_D$; the other is symmetric. By induction, $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi_1)\cup\mathcal{V}} \models \varphi_1$

and $\nu_{\text{out}} \supseteq \nu_{\text{in}}$. Again, $\mathit{fvars}(\varphi_1) \cup \mathcal{V} = \mathit{fvars}(\varphi) \cup \mathcal{V}$ from the conditions on $\mathcal{V}$, and we are done.

Existential Quantification. If φ is $\exists x\, \varphi_1$, then without loss of generality we may assume that $x \notin V$. By induction, we have an expression α_1 for φ_1 and $\mathcal{V}$. We show that this expression also works for φ. Take ν_{in} to be an arbitrary valuation on $\mathcal{V}$, and ν'_{in} to be any valuation such that $\nu'_{\text{in}} \supseteq \nu_{\text{in}}$.

We first prove completeness. Let ν be a valuation on $\mathit{fvars}(\varphi) \cup \mathcal{V}$ such that $\nu_{\text{in}} \subseteq \nu$. Now, suppose that $D, \nu \models \varphi$, and hence, $D, \nu \cup \nu_x \models \varphi_1$ where ν_x is a valuation on $\{x\}$. We want to verify that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha_1]\!]_D$ and $\nu_{\text{out}}|_{\mathit{fvars}(\varphi) \cup \mathcal{V}} = \nu$. By induction, we know that such ν_{out} exists but with $\nu_{\text{out}}|_{\mathit{fvars}(\varphi_1) \cup \mathcal{V}} = \nu \cup \nu_x$. Since x belongs neither to $\mathit{fvars}(\varphi)$ nor to $\mathcal{V}$, we easily obtain $\nu_{\text{out}}|_{\mathit{fvars}(\varphi) \cup \mathcal{V}} = \nu$, and we are done.

To show soundness, let $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha_1]\!]_D$. By induction, $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi_1) \cup \mathcal{V}} \models \varphi_1$, so certainly $D, \nu_{\text{out}}|_{\mathit{fvars}(\varphi) \cup \mathcal{V}} \models \varphi$. What remains to show is that $\nu_{\text{out}} \supseteq \nu_{\text{in}}$ which is clear from the induction step.

Negation. Finally, if φ is $\neg\varphi_1$, then by induction we have an expression α_1 for φ_1 and $\mathcal{V}$. Fix an arbitrary constant c, and a fresh variable u. Let Z denote $(\mathit{vars}(\alpha_1) - \mathcal{V}) \cup \{u\}$, and let ξ be the composition of all expressions $(z := c)$ for $z \in Z$. We show that α can be taken to be $\xi \;-\; \alpha_1 \,;\, \xi$. Note that $\mathit{fvars}(\varphi) = \mathit{fvars}(\varphi_1) \subseteq \mathcal{V}$ (by the $\mathcal{V}$-executability of φ).

We first prove completeness. Suppose that $D, \nu_{\text{in}} \models \varphi$. We want to verify that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$ and $\nu_{\text{out}}|_{\mathcal{V}} = \nu_{\text{in}}$. Take ν'_{out} to be the valuation that agrees with ν'_{in} outside Z (and hence, $\nu'_{\text{out}}|_{\mathcal{V}} = \nu_{\text{in}}$), and moreover, it assigns the value c for every $z \in Z$. It is clear that $(\nu'_{\text{in}}, \nu'_{\text{out}}) \in [\![\xi]\!]_D$. For the sake of contradiction, suppose $(\nu'_{\text{in}}, \nu'_{\text{out}}) \in [\![\alpha_1 \,;\, \xi]\!]_D$. Then, by definition, there exists ν such that $(\nu'_{\text{in}}, \nu) \in [\![\alpha_1]\!]_D$. By induction, we know that $D, \nu|_{\mathcal{V}} \models \varphi_1$ and $\nu|_{\mathcal{V}} = \nu_{\text{in}}$. It follows that $D, \nu_{\text{in}} \not\models \varphi$, which is a contradiction. Thus, $(\nu'_{\text{in}}, \nu'_{\text{out}}) \notin [\![\alpha_1 \,;\, \xi]\!]_D$, whence, $(\nu'_{\text{in}}, \nu'_{\text{out}}) \in [\![\alpha]\!]_D$, as desired.

To show soundness, suppose that there exists a valuation ν_{out} such that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\alpha]\!]_D$. We want to verify that $D, \nu_{\text{in}} \models \varphi$. By the semantics of α, we obtain that $(\nu'_{\text{in}}, \nu_{\text{out}}) \in [\![\xi]\!]_D$, and $(\nu'_{\text{in}}, \nu_{\text{out}}) \notin [\![\alpha_1 \,;\, \xi]\!]_D$. From the former, we obtain that $\nu_{\text{out}} = \nu'_{\text{in}}$ outside Z which is disjoint from $\mathcal{V}$. For the sake of contradiction, assume that $D, \nu_{\text{in}} \models \varphi_1$. Then, by induction, there is a ν'_{out} such that $(\nu'_{\text{in}}, \nu'_{\text{out}}) \in [\![\alpha_1]\!]_D$ and $\nu'_{\text{out}} = \nu'_{\text{in}} = \nu_{\text{out}}$ on $\mathcal{V}$. What remains to show is that $(\nu'_{\text{out}}, \nu_{\text{out}}) \in [\![\xi]\!]_D$ yielding the contradiction. It is not hard to see that $\nu'_{\text{in}} = \nu'_{\text{out}}$ outside $\mathit{vars}(\alpha_1) - \mathcal{V}$ which contains all the set of variables outside Z. Thus, $\nu_{\text{out}} = \nu'_{\text{out}}$ outside Z, whence, $(\nu'_{\text{out}}, \nu_{\text{out}}) \in [\![\xi]\!]_D$ as desired. □

6.2. **From FLIF to io-disjoint FLIF.** We prove that β constructed by the method described in Section 5.1.4 satisfies the statement of Lemma 5.3. The base cases are straightforwardly verified. For every inductive case, we need to verify several things:

Inputs: $I(\beta) = I(\alpha)$.

io-disjointness: Every subexpression of β, including β itself, must have disjoint inputs and outputs.

Outputs: $O(\beta) \supseteq \rho(O(\alpha))$.

No clashes: $O(\beta) - \rho(O(\alpha))$ is disjoint from W.

Completeness: For any instance D and $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, we want to find ν such that $(\nu_1, \nu) \in [\![\beta]\!]_D$ and $\nu(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$.

Soundness: For any $(\nu_1, \nu_2) \in [\![\beta]\!]_D$, we want to find ν such that $(\nu_1, \nu) \in [\![\alpha]\!]_D$ and $\nu(y) = \nu_2(\rho(y))$ for $y \in O(\alpha)$.

6.2.1. *Composition.* Henceforth, for any expression δ, we will use the notation

$$\nu_1 \xrightarrow{\delta} \nu_2$$

to indicate that $(\nu_1, \nu_2) \in [\![\delta]\!]_D$.

Inputs. We first analyze inputs and outputs for $\theta(\beta_2)$. Inputs pose no difficulty (note that $I(\beta_2) = I(\alpha_2)$). As to outputs, θ only changes variables in $I(\alpha_2)$ and β_2 is io-disjoint by induction, so θ has no effect on $O(\beta_2)$. Hence:

$$\begin{aligned} I(\theta(\beta_2)) &= (I(\alpha_2) - O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1)) \\ O(\theta(\beta_2)) &= O(\beta_2) \end{aligned}$$

Calculating $I(\beta)$, the part of $I(\theta(\beta_2))$ that is contained in $\rho_1(O(\alpha_1))$ disappears, because $\rho_1(O(\alpha_1))$ is contained in $O(\beta_1)$. Also, $I(\beta_1) = I(\alpha_1)$ by induction. Thus $I(\beta) = I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1)) = I(\alpha)$ as desired.

Outputs. We verify:

$$\begin{aligned} \rho(O(\alpha)) &= \rho(O(\alpha_1)) \cup \rho(O(\alpha_2)) \\ &= \rho(O(\alpha_1) - O(\alpha_2)) \cup \rho(O(\alpha_2)) \\ &= \rho_1(O(\alpha_1) - O(\alpha_2)) \cup \rho_2(O(\alpha_2)) \\ &\subseteq O(\beta_1) \cup O(\beta_2) \\ &= O(\beta). \end{aligned}$$

io-disjointness. Expression β is io-disjoint since $O(\beta_1)$ and $O(\beta_2)$ are disjoint from $vars(\alpha)$ by construction. For subexpression $\theta(\beta_2)$, recall $I(\theta(\beta_2))$ and $O(\theta(\beta_2))$ as calculated above. The part contained in $I(\alpha_2)$ is disjoint from $O(\beta_2)$ since $I(\alpha_2) = I(\beta_2)$ and β_2 is io-disjoint by induction. We write the other part as $\rho_1(I(\alpha_2) \cap O(\alpha_1) \cap O(\alpha_2)) \cup \rho((I(\alpha_2) \cap O(\alpha_1)) - O(\alpha_2))$. The first term is disjoint from $O(\beta_2)$ by definition of ρ_1.

The second term is dealt with by the more general claim that $\rho(O(\alpha_1) - O(\alpha_2))$ *is disjoint from* $O(\beta_2)$. Towards a proof, let $y \in O(\alpha_1) - O(\alpha_2)$ and assume for the sake of contradiction that $\rho(y) \in O(\beta_2)$. Then $\rho(y) \in O(\beta_2) - \rho(O(\alpha_2))$, which by induction is disjoint from W_2, which includes $\rho(O(\alpha_1))$. However, since $y \in O(\alpha_1)$, this is a contradiction.

No clashes. We have

$$\begin{aligned} O(\beta) - \rho(O(\alpha)) &= (O(\beta_1) \cup O(\beta_2)) - \rho(O(\alpha_1) \cup O(\alpha_2)) \\ &\subseteq (O(\beta_1) - \rho(O(\alpha_1)) \cup (O(\beta_2) - \rho(O(\alpha_2)). \end{aligned}$$

By induction, the latter two terms are disjoint from $W_1 \supseteq W$ and $W_2 \supseteq W$, respectively.

Completeness. Since $(\nu_1, \nu_2) \in [\![\alpha]\!]_D$, there exists ν such that

$$\nu_1 \xrightarrow{\alpha_1} \nu \xrightarrow{\alpha_2} \nu_2.$$

By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\beta_1]\!]_D$ and $\nu_3(\rho_1(y)) = \nu(y)$ for $y \in O(\alpha_1)$. Also by induction, there exists ν_4 such that $(\nu, \nu_4) \in [\![\beta_2]\!]_D$ and $\nu_4(\rho_2(y)) = \nu_2(y)$ for $y \in O(\alpha_2)$. By the Renaming Lemma (5.2), we have $(\nu \circ \theta, \nu_4 \circ \theta) \in [\![\theta(\beta_2)]\!]_D$.

We claim that ν_3 agrees with $\nu \circ \theta$ on $I(\theta(\beta))$. Recalling that the latter equals $(I(\alpha_2) - O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$, we verify this claim as follows.

- We begin by verifying that θ is the identity on $I(\alpha_2) - O(\alpha_1)$. Indeed, let $u \in I(\alpha_2) - O(\alpha_1)$. Note that θ is the identity outside $(I(\alpha_2) \cap O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$. Clearly u does not belong to the first term. Also u does not belong to the second term, since the image of ρ_1 is disjoint from $vars(\alpha)$.
- Now let $u \in I(\alpha_2) - O(\alpha_1)$. Then $\theta(u) = u$, so $(\nu \circ \theta)(u) = \nu(u)$. Since $\nu_1 \xrightarrow{\alpha_1} \nu$ and u does not belong to $O(\alpha_1)$, we have $\nu(u) = \nu_1(u)$. Also, $\nu_1 \xrightarrow{\beta_1} \nu_3$ and u does not belongs to $O(\beta_1)$ since $O(\beta_1)$ is disjoint from $vars(\alpha)$. Hence, $\nu_1(u) = \nu_3(u)$ so we get $\nu(u) = \nu_3(u)$.
- Let $u \in I(\alpha_2) \cap O(\alpha_1)$. Then $(\nu \circ \theta)(\rho_1(u)) = \nu(\theta(\theta(u))) = \nu(u)$. The latter equals $\nu_3(\rho_1(u))$ by definition of ν_3.

We can now apply input-output determinacy and obtain ν_5 such that $(\nu_3, \nu_5) \in [\![\theta(\beta_2)]\!]_D$ and ν_5 agrees with $\nu_4 \circ \theta$ on $O(\beta_2)$. It follows that $(\nu_1, \nu_5) \in [\![\beta]\!]_D$, so we are done if we can show that $\nu_5(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$. We distinguish two cases.

First, assume $y \in O(\alpha_2)$. Then $\nu_5(\rho(y)) = \nu_5(\rho_2(y)) = (\nu_4 \circ \theta)(\rho_2(y))$ by definition of ν_5. Now observe that $\theta(\rho_2(y)) = \rho_2(y)$. Indeed, $\rho_2(y)$ belongs to $O(\beta_2)$, while θ is the identity outside $(I(\alpha_2) \cap O(\alpha_1)) \cup \rho_1(I(\alpha_2) \cap O(\alpha_1))$. The first term is disjoint from $O(\beta_2)$ since $O(\beta_2)$ is disjoint from $vars(\alpha)$. The second term is disjoint from $O(\beta_2)$ as already shown in the io-disjointness proof. So, we obtain $\nu_4(\rho_2(y))$, which equals $\nu_2(y)$ by definition of ν_4.

Second, assume $y \in O(\alpha_1) - O(\alpha_2)$. Since $(\nu_3, \nu_5) \in [\![\theta(\beta_2)]\!]_D$ and $O(\theta(\beta_2)) = O(\beta_2)$ is disjoint from $\rho(O(\alpha_1) - O(\alpha_2))$ as seen in the disjointness proof, $\nu_5(\rho(y)) = \nu_3(\rho(y))$. Since $y \notin O(\alpha_2)$, we have $\nu_3(\rho(y)) = \nu_3(\rho_1(y))$, which equals $\nu(y)$ by definition of ν_3. Now $\nu(y) = \nu_2(y)$ since $(\nu, \nu_2) \in [\![\alpha_2]\!]_D$ and $y \notin O(\alpha_2)$.

Soundness. The proof for soundness is remarkably symmetrical to that for completeness. Such symmetry is not present in the proofs for the other operators. We cannot yet explain well why the symmetry is present onlu for composition.

Since $(\nu_1, \nu_2) \in [\![\beta]\!]_D$, there exists ν such that

$$\nu_1 \xrightarrow{\beta_1} \nu \xrightarrow{\theta(\beta_2)} \nu_2.$$

By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\alpha_1]\!]_D$ and $\nu_3(y) = \nu(\rho_1(y))$ for $y \in O(\alpha_1)$. By the Renaming Lemma, we have $(\nu \circ \theta, \nu_2 \circ \theta) \in [\![\beta_2]\!]_D$ (note that $\theta^{-1} = \theta$). By induction, there exists ν_4 such that $(\nu \circ \theta, \nu_4) \in [\![\alpha_2]\!]_D$ and $\nu_4(y) = (\nu_2 \circ \theta)(\rho_2(y))$ for $y \in O(\alpha_2)$.

Using analogous reasoning as in the completeness proof, it can be verified that ν_3 agrees with $\nu \circ \theta$ on $I(\alpha_2)$. Hence, by input-output determinacy, there exists ν_5 such that $(\nu_3, \nu_5) \in [\![\alpha_2]\!]_D$ and ν_5 agrees with ν_4 on $O(\alpha_2)$. It follows that $(\nu_1, \nu_5) \in [\![\alpha]\!]_D$, so we are done if we can show that $\nu_5(y) = \nu_2(\rho(y))$ for $y \in O(\alpha)$. This is shown by analogous reasoning as in the completeness proof.

6.2.2. *Union.*
Inputs. Let $\{i, j\} = \{1, 2\}$. We begin by noting:

$$I(\gamma_i) = O(\alpha_j) - O(\alpha_i)$$
$$O(\gamma_i) = \rho(O(\alpha_j) - O(\alpha_i))$$

Note that $I(\gamma_i)$, being a subset of $vars(\alpha)$, is disjoint from $O(\beta_i)$, so $I(\beta_i\,;\gamma_i)$ is simply $I(\beta_i) \cup I(\gamma_i)$. By induction, $I(\beta_i) = I(\alpha_i)$ and $O(\beta_i)$ contains $\rho(O(\alpha_i))$. Hence:

$$I(\beta_i\,;\gamma_i) = I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i))$$
$$O(\beta_i\,;\gamma_i) = O(\beta_i) \cup \rho(O(\alpha_j))$$

We next analyze η_i. Recall that this expression was defined by two cases.

(a) If $O(\beta_i)$ is nonempty, $I(\eta_i) \subseteq O(\beta_i)$.
(b) Otherwise, $I(\eta_i) \subseteq vars(\alpha_j)$. However, if $O(\beta_i)$ is empty then $O(\alpha_i)$ is too, so that $I(\alpha) = I(\alpha_i) \cup I(\alpha_j) \cup O(\alpha_j) = I(\alpha_i) \cup vars(\alpha_j)$. Hence, in this case, $I(\eta_i) \subseteq I(\alpha)$.

The output is the same in both cases:

$$O(\eta_i) = O(\beta_j) - \rho(O(\alpha_j))$$

Composing $\beta_i\,;\gamma_i$ with η_i, we continue with the two above cases.

(a) In this case $I(\eta_i)$ is contained in $O(\beta_i\,;\gamma_i)$, so $I(\beta_i\,;\gamma_i\,;\eta_i) = I(\beta_i\,;\gamma_i)$.
(b) In this case $I(\eta_i)$ is disjoint from $O(\beta_i\,;\gamma_i)$, and $I(\beta_i\,;\gamma_i\,;\eta_i)$ equals $I(\beta_i\,;\gamma_i)$ to which some element of $I(\alpha)$ is added.

In both cases, we can state that

$$I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i)) \subseteq I(\beta_i\,;\gamma_i\,;\eta_1) \subseteq I(\alpha).$$

For outputs, we have

$$O(\beta_i\,;\gamma_i\,;\eta_i) = O(\beta_1) \cup O(\beta_2).$$

The set of inputs of the final expression $\beta = (\beta_1\,;\gamma_1\,;\eta_1) \cup (\beta_2\,;\gamma_2\,;\eta_2)$ equals the union of inputs of the two top-level subexpressions, since these two subexpressions have the same outputs $(O(\beta_1) \cup O(\beta_2))$. Hence

$$I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2)) \subseteq I(\beta) \subseteq I(\alpha).$$

Since the left expression equals $I(\alpha)$ by definition, we obtain that $I(\beta) = I(\alpha)$ as desired.

Outputs. From the above we have $O(\beta) = O(\beta_1) \cup O(\beta_2)$. Since $O(\beta_i) \supseteq \rho(O(\alpha_i))$ by induction, we obtain $O(\beta) \supseteq \rho(O(\alpha_1) \cup O(\alpha_2)) = \rho(O(\alpha))$ as desired.

io-disjointness. Let $i = 1, 2$. Expression γ_i is io-disjoint since the image of ρ is disjoint from $vars(\alpha)$. Then $\beta_i\,;\gamma_i$ is io-disjoint because both $O(\beta_i)$ and the image of ρ are disjoint from $vars(\alpha)$. For the same reason, $\beta_i\,;\gamma_i\,;\eta_i$ and β are io-disjoint. We still need to look at η_i. In case (b), $I(\eta_i) \subseteq I(\alpha)$ so io-disjointness follows again because $O(\beta_j)$ is disjoint from $vars(\alpha)$. In case (a), we look at $i = 1$ and $i = 2$ separately. For $i = 1$ we observe that $O(\eta_1) = O(\beta_2) - \rho(O(\alpha_2))$ is disjoint from W_2, which includes $O(\beta_1)$. For $i = 2$ we write $O(\beta_2) = \rho(O(\alpha_2)) \cup (O(\beta_2) - \rho(O(\alpha_2)))$. The first term is disjoint from $O(\eta_2) = O(\beta_1) - \rho(O(\alpha_1))$ since the latter is disjoint from W_1 which includes $\rho(O(\alpha_2))$. The second term is disjoint from $O(\beta_1)$ as we have just seen.

No clashes. We verify:

$$\begin{aligned} O(\beta) - \rho(O(\alpha)) &= (O(\beta_1) \cup O(\beta_2)) - \rho(O(\alpha_1) \cup O(\alpha_2)) \\ &\subseteq (O(\beta_1) - \rho(O(\alpha_1))) \cup (O(\beta_2) - \rho(O(\alpha_2))). \end{aligned}$$

By induction, both of the latter terms are disjoint from W, which confirms that there are no clashes.

Completeness. Assume $(\nu_1, \nu_2) \in [\![\alpha_1]\!]_D$; the reasoning for α_2 is analogous. By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\beta_1]\!]_D$ and $\nu_3(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha_1)$.

Note that each of the expressions γ_i and η_i for $i = 1, 2$ is a composition of variable assignments. For any such expression δ and any valuation ν there always exists a unique ν' such that $(\nu, \nu') \in [\![\delta]\!]_D$ (even independently of D).

Now let

$$\nu_3 \xrightarrow{\gamma_1} \nu_4 \xrightarrow{\eta_1} \nu_5,$$

so that $(\nu_1, \nu_5) \in [\![\beta]\!]_D$. If we can show that $\nu_5(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$ we are done. Thereto, first note that η_1 does not change variables in $\rho(O(\alpha))$. Indeed, for $\rho(O(\alpha_2))$ this is obvious from $O(\eta_1) = O(\beta_2) - \rho(O(\alpha_2))$; for $\rho(O(\alpha_1))$ this follows because by induction, $O(\beta_2) - \rho(O(\alpha_2))$ is disjoint from W_2, which includes $O(\beta_1)$, which includes $\rho(O(\alpha_1))$. So, by $\nu_4 \xrightarrow{\eta_1} \nu_5$ we are down to showing that $\nu_4(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$. We distinguish two cases.

If $y \in O(\alpha_1)$, since $\nu_3 \xrightarrow{\gamma_1} \nu_4$ and γ_1 does not change variables in $\rho(O(\alpha_1))$, we have $\nu_4(\rho(y)) = \nu_3(\rho(y))$, which equals $\nu_2(y)$ by definition of ν_3.

If $y \in O(\alpha_2) - O(\alpha_1)$, then $\nu_4(\rho(y)) = \nu_3(y)$ by $\nu_3 \xrightarrow{\gamma_1} \nu_4$. Now since

$$\nu_3 \xleftarrow{\beta_1} \nu_1 \xrightarrow{\alpha_1} \nu_2$$

and $y \notin O(\beta_1) \cup O(\alpha_1)$, we get $\nu_3(y) = \nu_2(y)$ as desired. (The reason for $y \notin O(\beta_1)$ is that by induction, $O(\beta_1)$ is disjoint from W_1 which includes $vars(\alpha)$.)

Soundness. Assume $(\nu_1, \nu_2) \in [\![\beta_1 ; \gamma_1 ; \eta_1]\!]_D$; the reasoning for $\beta_2 ; \gamma_2 ; \eta_2$ is analogous. Then there exist ν_3 and ν_4 such that

$$\nu_1 \xrightarrow{\beta_1} \nu_3 \xrightarrow{\gamma_1} \nu_4 \xrightarrow{\eta_1} \nu_2. \qquad (*)$$

By induction, there exists ν such that $(\nu_1, \nu) \in [\![\alpha_1]\!]_D \subseteq [\![\alpha]\!]_D$ and $\nu(y) = \nu_3(\rho(y))$ for $y \in O(\alpha_1)$. As observed in the completeness proof, γ_1 and η_1 do not touch variables in $\rho(O(\alpha_1))$. Since $(*)$ shows that γ_1 followed by η_1 maps ν_3 to ν_2, also $\nu(y) = \nu_2(\rho(y))$ for $y \in O(\alpha_1)$.

If we can show the same for $y \in O(\alpha_2) - O(\alpha_1)$, we have covered all $y \in O(\alpha)$ and we are done. This is verified as follows. By inertia, we have $\nu(y) = \nu_1(y) = \nu_3(y)$, the latter equality because $O(\beta_1)$ is disjoint from $vars(\alpha)$. From $\nu_3 \xrightarrow{\gamma_1} \nu_4$ we have $\nu_3(y) = \nu_4(\rho(y))$. Now the latter equals $\nu_2(\rho(y))$ since $\nu_4 \xrightarrow{\eta_1} \nu_2$ and η_1 does not touch variables in $\rho(O(\alpha_2))$.

6.2.3. *Difference.*
Inputs. Let $\{i, j\} = \{1, 2\}$. We begin by noting:

$$I(\gamma_i) = O(\alpha_j) - O(\alpha_i)$$
$$O(\gamma_i) = \rho_j(O(\alpha_j) - O(\alpha_i))$$

Slightly adapting the calculation of inputs in the proof for union (Section 6.2.2), we next note:

$$I(\beta_i \,;\gamma_i) = I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i))$$
$$O(\beta_i \,;\gamma_i) = O(\beta_i) \cup \rho_j(O(\alpha_j) - O(\alpha_i))$$

We next analyze η_1. Recall that this expression was defined by two cases.

(a) If $O(\alpha_1)$ and $O(\alpha_2)$ intersect, $I(\eta_1) \subseteq \rho(O(\alpha_1) \cap O(\alpha_2))$.
(b) Otherwise, $I(\eta_1) \subseteq \mathit{vars}(\alpha_2)$. However, note in this case that $I(\alpha) = \mathit{vars}(\alpha)$, so that $I(\eta_1) \subseteq I(\alpha)$.

Regardless of the case,

$$O(\eta_1) = O(\beta_2) - \rho_2(O(\alpha_2)).$$

Composing $\beta_i \,;\gamma_i$ with η_1, we continue with the above two cases.

(a) By induction, $O(\beta_i)$ contains $\rho_i(O(\alpha_i))$, and ρ agrees ρ_i on $O(\alpha_1) \cap O(\alpha_2)$. Hence $I(\eta_1) \subseteq O(\beta_i \,;\gamma_i)$ and thus

$$I(\beta_i \,;\gamma_i \,;\eta_1) = I(\beta_i \,;\gamma_i) = I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i)).$$

(b) In this case $I(\eta_1) \subseteq I(\alpha)$ which is disjoint from $O(\beta_i \,;\gamma_i)$. Note that also $I(\beta_i \,;\gamma_i) \subseteq I(\alpha)$.

In both cases, we can state that

$$I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i)) \subseteq I(\beta_i \,;\gamma_i \,;\eta_1) \subseteq I(\alpha).$$

For outputs, we have

$$O(\beta_i \,;\gamma_i \,;\eta_1) = O(\beta_i) \cup \rho_j(O(\alpha_j) - O(\alpha_i)) \cup (O(\beta_2) - \rho_2(O(\alpha_2))).$$

Composing further with η_2, which is defined similarly to η_1, we can reason similarly and still state that

$$I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i)) \subseteq I(\beta_i \,;\gamma_i \,;\eta_1 \,;\eta_2) \subseteq I(\alpha).$$

For outputs, note that $O(\eta_2) = O(\beta_1) - \rho_1(O(\alpha_1))$. Uniting this to the expression for $O(\beta_i \,;\gamma_i \,;\eta_1)$ above, we obtain

$$O(\beta_i \,;\gamma_i \,;\eta_1 \,;\eta_2) = O(\beta_1) \cup O(\beta_2).$$

Indeed, the only part of $O(\beta_1) \cup O(\beta_2)$ that is not obviously there is $\rho_j(O(\alpha_j) \cap O(\alpha_i))$. However, that part is contained in $\rho(O(\alpha_i))$, because ρ_j agrees with ρ on $O(\alpha_1) \cap O(\alpha_2)$. Since $\rho(O(\alpha_i)) \subseteq O(\beta_i)$, the part is included after all.

With the above results we can reason exactly as in the proof for union and obtain that $I(\beta) = I(\alpha)$ as desired.

Outputs. From the above we have $O(\beta) = O(\beta_1) \cup O(\beta_2)$. Since $O(\beta_1) \supseteq \rho_1(O(\alpha_1))$ by induction, and $\rho_1 = \rho$ and $O(\alpha) = O(\alpha_1)$, we obtain $O(\beta) \supseteq \rho(O(\alpha))$ as desired.

io-disjointness. Let $i = 1, 2$. Expression γ_i is io-disjoint by the choice of ρ_j. Then $\beta_i \,;\, \gamma_i$ is io-disjoint because both $O(\beta_i)$ and the image of ρ_j are disjoint from $vars(\alpha)$. Regarding η_1, we have seen that either (a) $I(\eta_1) \subseteq \rho(O(\alpha_1) \cap O(\alpha_2)) \subseteq \rho_2(O(\alpha_2))$, or (b) $I(\eta_1) \subseteq I(\alpha)$. In case (a) $I(\eta_1)$ is clearly disjoint from $O(\eta_1) = O(\beta_2) - \rho_2(O(\alpha_2))$. Also in case (b) η_1 is io-disjoint because $O(\beta_2)$ is disjoint from $vars(\alpha)$. Using similar reasoning, the expressions $\beta_i \,;\, \gamma_i \,;\, \eta_1$, η_2, $\beta_i \,;\, \gamma_i \,;\, \eta_1 \,;\, \eta_2$, and finally β, are seen to be io-disjoint.

No clashes. Note that $\rho(O(\alpha)) = \rho_1(O(\alpha_1))$, and recall that $O(\beta) = O(\beta_1) \cup O(\beta_2)$. Hence we can write $O(\beta) - \rho(O(\alpha))$ as

$$(O(\beta_1) - \rho_1(O(\alpha_1))) \cup (O(\beta_2) - \rho_1(O(\alpha_1))).$$

The first term is disjoint from W by construction and induction. For the second term, note that $O(\beta_2)$ can be written as a disjoint union

$$\rho_2(O(\alpha_2) \cap O(\alpha_1)) \cup \rho_2(O(\alpha_2) - O(\alpha_1)) \cup (O(\beta_2) - \rho_2(O(\alpha_2))).$$

Again by construction and induction, the second and third terms are disjoint from W_2, which includes $O(\beta_1)$, which includes $\rho_1(O(\alpha_1))$. On the other hand, the first term is included in $\rho_1(O(\alpha_1))$ since ρ_1 and ρ_2 agree on $O(\alpha_1) \cap O(\alpha_2)$. Hence, $O(\beta_2) - \rho_1(O(\alpha_1))$ reduces to the union of the second and third terms, which are disjoint from W_2, which includes W, as desired.

Completeness. Since $(\nu_1, \nu_2) \in [\![\alpha_1 - \alpha_2]\!]_D$, in particular $(\nu_1, \nu_2) \in [\![\alpha_1]\!]_D$, so by induction there exists ν_3 such that $(\nu_1, \nu_3) \in [\![\beta_1]\!]_D$ and $\nu_3(\rho_1(y)) = \nu_2(y)$ for $y \in O(\alpha_1)$.

Recall the output variables of γ_i and η_i for $i = 1, 2$:

$$O(\gamma_1) = \rho_2(O(\alpha_2) - O(\alpha_1))$$
$$O(\gamma_2) = \rho_1(O(\alpha_1) - O(\alpha_2))$$
$$O(\eta_1) = O(\beta_2) - \rho_2(O(\alpha_2))$$
$$O(\eta_2) = O(\beta_1) - \rho_1(O(\alpha_1))$$

We observe:

(1) *None of the assignments in γ_1, η_1 or η_2 affect variables in $\rho_1(O(\alpha_1))$.*
This claim is clear for η_2. For γ_1 it holds since ρ_2 was chosen such that its image on $O(\alpha_2) - O(\alpha_1)$ is disjoint from W_2, which includes $O(\beta_1)$, which includes $\rho_1(O(\alpha_1))$. For η_1 the claim holds because, by induction, $O(\eta_1)$ is again disjoint from W_2.

(2) *None of the assignments in γ_2, η_1 or η_2 affect variables in $\rho_2(O(\alpha_2))$.*
This claim is clear for η_1. Next consider γ_2. On $O(\alpha_2) - O(\alpha_1))$, we just noted that the image of ρ_2 is disjoint from $\rho_1(O(\alpha_1))$. Now let $y \in O(\alpha_2) \cap O(\alpha_1)$. Then $\rho_2(y) = \rho_1(y)$ and clearly $\rho_1(y) \notin \rho_1(O(\alpha_1) - O(\alpha_2))$. Finally, consider η_2. On $O(\alpha_2) - O(\alpha_1)$, we again use that the image of ρ_2 is disjoint from $O(\beta_1)$. On $O(\alpha_1) \cap O(\alpha_2)$, again the image of ρ_2 agrees with the image of ρ_1, which clearly is disjoint from $O(\eta_2)$.

Now, using the notation introduced in the completeness proof for union (Section 6.2.2), let

$$\nu_3 \xrightarrow{\gamma_1} \nu_4 \xrightarrow{\eta_1} \nu_5 \xrightarrow{\eta_2} \nu_6$$

so that $(\nu_1, \nu_6) \in [\![\beta_1 \,;\, \gamma_1 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. By Observation (1), for $y \in O(\alpha) = O(\alpha_1)$, we still have $\nu_6(\rho_1(y)) = \nu_3(\rho_1(y)) = \nu_2(y)$. Thus, completeness is proved provided we can show that $(\nu_1, \nu_6) \notin [\![\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2]\!]_D$.

For the sake of contradiction, assume $(\nu_1, \nu_6) \in [\![\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. By the identity property (Proposition 4.11), also $(\nu_6, \nu_6) \in [\![\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. Hence, there exists ν_7 such that $(\nu_6, \nu_7) \in [\![\beta_2]\!]_D$ and $(\nu_7, \nu_6) \in [\![\gamma_2 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. By inertia, ν_6 and ν_7 can differ only on $O(\beta_2)$, and among γ_2, η_1 and η_2, only η_1 can change variables in $O(\beta_2)$. Hence we have

$$\nu_7 \xrightarrow{\gamma_2} \nu_7 \xrightarrow{\eta_1} \nu_6 \xrightarrow{\eta_2} \nu_6.$$

Since $(\nu_6, \nu_7) \in [\![\beta_2]\!]_D$, by induction there exists ν_8 such that $(\nu_6, \nu_8) \in [\![\alpha_2]\!]_D$ and $\nu_8(y) = \nu_7(\rho_2(y))$ for $y \in O(\alpha_2)$. Recall that $(\nu_1, \nu_6) \in [\![\beta_1 \,;\, \gamma_1 \,;\, \eta_1 \,;\, \eta_2]\!]_D$, so ν_1 and ν_6 agree outside $O(\beta_1) \cup O(\beta_2)$, which is disjoint from $vars(\alpha)$ which includes $I(\alpha_2)$. Hence we can apply input-output determinacy, yielding a valuation ν such that $(\nu_1, \nu) \in [\![\alpha_2]\!]_D$ and ν agrees with ν_8 on $O(\alpha_2)$. If we can show that $\nu = \nu_2$ we have arrived at a contradiction, since $(\nu_1, \nu_2) \notin [\![\alpha_2]\!]_D$.

By inertia, ν and ν_1 agree outside $O(\alpha_2)$, and ν_1 and ν_2 agree outside $O(\alpha_1)$. Thus ν and ν_2 already agree outside $O(\alpha_1) \cup O(\alpha_2)$ and we can focus on that set of variables. We distinguish three cases.

First, let $y \in O(\alpha_1) \cap O(\alpha_2)$. Note that $\rho(y) = \rho_1(y) = \rho_2(y)$. By definition of ν and ν_8, we have $\nu(y) = \nu_8(y) = \nu_7(\rho(y))$. Since

$$\nu_3 \xrightarrow{\gamma_1} \nu_4 \xrightarrow{\eta_1} \nu_5 \xrightarrow{\eta_2} \nu_6 \xleftarrow{\eta_1} \nu_7, \tag{6.1}$$

by Observation (1), we have $\nu_7(\rho(y)) = \nu_3(\rho(y))$. The latter indeed equals $\nu_2(y)$, by definition of ν_3.

Second, let $y \in O(\alpha_2) - O(\alpha_1)$. As before we have $\nu(y) = \nu_7(\rho_2(y))$. By (6.1) and Observation (2), $\nu_7(\rho_2(y)) = \nu_4(\rho_2(y))$. The latter equals $\nu_3(y)$ since $\nu_3 \xrightarrow{\gamma_1} \nu_4$. Now since $(\nu_1, \nu_3) \in [\![\beta_1]\!]_D$ and $(\nu_1, \nu_2) \in [\![\alpha_1]\!]_D$ and y is neither in $O(\beta_1)$ nor in $O(\alpha_1)$, we get $\nu_3(y) = \nu_1(y) = \nu_2(y)$.

Third, let $y \in O(\alpha_1) - O(\alpha_2)$. Since $(\nu_1, \nu) \in [\![\alpha_2]\!]_D$ and $y \notin O(\alpha_2)$, by inertia $\nu(y) = \nu_1(y)$. Likewise, since

$$\nu_1 \xrightarrow{\beta_1;\gamma_1;\eta_1;\eta_2} \nu_6 \xrightarrow{\beta_2} \nu_7$$

and $y \notin O(\beta_1) \cup O(\beta_2)$, we get $\nu_1(y) = \nu_6(y) = \nu_7(y)$. Since $\nu_7 \xrightarrow{\gamma_2} \nu_7$ we have $\nu_7(y) = \nu_7(\rho_1(y))$. In the first case we already noted that $\nu_7(\rho_1(y)) = \nu_3(\rho_1(y))$. Now the latter equals $\nu_2(y)$ by definition of ν_3, and we are done.

Soundness. Since $(\nu_1, \nu_2) \in [\![\beta]\!]_D$, we have $(\nu_1, \nu_2) \in [\![\beta_1 \,;\, \gamma_1 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. By the identity property, also $(\nu_2, \nu_2) \in [\![\beta_1 \,;\, \gamma_1 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. Hence there exists ν_3 such that $(\nu_2, \nu_3) \in [\![\beta_1]\!]_D$ and $(\nu_3, \nu_2) \in [\![\gamma_1 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. By inertia, ν_2 and ν_3 can differ only on $O(\beta_1)$, and among γ_1, η_1 and η_2, only η_2 can change variables in $O(\beta_1)$. Hence we have

$$\nu_3 \xrightarrow{\gamma_1} \nu_3 \xrightarrow{\eta_1} \nu_3 \xrightarrow{\eta_2} \nu_2.$$

Since $(\nu_2, \nu_3) \in [\![\beta_1]\!]_D$, by induction there exists ν_4 such that $(\nu_2, \nu_4) \in [\![\alpha_1]\!]_D$ and $\nu_4(y) = \nu_3(\rho_1(y))$ for $y \in O(\alpha_1)$. Note that ν_1 and ν_2 agree outside $O(\beta)$ which is disjoint from $I(\alpha_1)$. Hence we can apply input-output determinacy, yielding a valuation ν such that $(\nu_1, \nu) \in [\![\alpha_1]\!]_D$ and ν agrees with ν_4 on $O(\alpha_1)$. Our goal is to show that $(\nu_1, \nu) \notin [\![\alpha_2]\!]_D$.

For the sake of contradiction, assume $(\nu_1, \nu) \in [\![\alpha_2]\!]_D$. Then by induction, there exists ν_5 such that $(\nu_1, \nu_5) \in [\![\beta_2]\!]_D$ and $\nu_5(\rho_2(y)) = \nu(y)$ for $y \in O(\alpha_2)$. Let

$$\nu_5 \xrightarrow{\gamma_2} \nu_6 \xrightarrow{\eta_1} \nu_7 \xrightarrow{\eta_2} \nu_8 \tag{6.2}$$

so that $(\nu_1, \nu_8) \in [\![\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2]\!]_D$. If we can show that $\nu_8 = \nu_2$, we have arrived at the desired contradiction since $(\nu_1, \nu_2) \notin [\![\beta_2 \,;\, \gamma_2 \,;\, \eta_1 \,;\, \eta_2]\!]_D$.

By inertia, ν_8 and ν_1, and ν_1 and ν_2, agree outside $O(\beta_1) \cup O(\beta_2)$. Thus ν_8 and ν_2 already agree outside $O(\beta_1) \cup O(\beta_2)$ and we can focus on that set of variables. Note that $O(\beta_1)$ contains $\rho_1(O(\alpha_1))$ and $O(\beta_2)$ contains $\rho_2(O(\alpha_2))$. Accordingly, we distinguish five cases.

(1) $\rho(O(\alpha_1) \cap O(\alpha_2))$. Let $y \in O(\alpha_1) \cap O(\alpha_2)$. By (6.2) and Observation (2), $\nu_8(\rho(y)) = \nu_5(\rho(y))$. By definition of ν_5, ν and ν_4 respectively, $\nu_5(\rho(y)) = \nu(y) = \nu_4(y) = \nu_3(\rho(y))$. The latter equals $\nu_2(\rho(y))$ since $\nu_3 \xrightarrow{\eta_2} \nu_2$.

(2) $\rho_2(O(\alpha_2) - O(\alpha_1))$. Let $y \in O(\alpha_2) - O(\alpha_1)$. As in case (1), $\nu_8(\rho_2(y)) = \nu(y)$. Since

$$\nu \xleftarrow{\alpha_1} \nu_1 \xrightarrow{\beta_1;\gamma_1;\eta_1;\eta_2} \nu_2 \xrightarrow{\beta_1} \nu_3$$

and $y \notin O(\alpha_1) \cup O(\beta_1) \cup O(\beta_2)$, we have $\nu(y) = \nu_3(y)$. The latter equals $\nu_3(\rho_2(y))$ by $\nu_3 \xrightarrow{\gamma_1} \nu_3$. Now by $\nu_3 \xrightarrow{\eta_2} \nu_2$ and Observation (2) we get $\nu_3(\rho_2(y)) = \nu_2(\rho_2(y))$.

(3) $\rho_1(O(\alpha_1) - O(\alpha_2))$. Let $y \in O(\alpha_1) - O(\alpha_2)$. By (6.2) and Observation (1), $\nu_8(\rho_1(y)) = \nu_6(\rho_1(y))$. The latter equals $\nu_5(y)$ by $\nu_5 \xrightarrow{\gamma_2} \nu_6$. Since

$$\nu_5 \xleftarrow{\beta_2} \nu_1 \xrightarrow{\alpha_2} \nu$$

and $y \notin O(\beta_2) \cup O(\alpha_2)$, we get $\nu_5(y) = \nu(y)$. The latter equals $\nu_2(\rho_1(y))$ as in case (1).

(4) $O(\beta_2) - \rho_2(O(\alpha_2))$. Let $y \in O(\beta_2) - \rho_2(O(\alpha_2))$. We distinguish two further cases following the definition of η_1, which involves the choice of a variable z.
   (a) $z = \rho(x)$ for some $x \in O(\alpha_1) \cap O(\alpha_2)$. Since $\nu_7 \xrightarrow{\eta_2} \nu_8$, we have $\nu_8(y) = \nu_7(\rho(x))$. The latter equals $\nu_3(\rho(x))$ as in case (1). Now $\nu_3 \xrightarrow{\eta_2} \nu_2$ yields $\nu_3(\rho(x)) = \nu_2(y)$.
   (b) In this case $z \in I(\alpha)$ (see the Inputs part of this proof). Since $\nu_7 \xrightarrow{\eta_2} \nu_8$, we have $\nu_8(y) = \nu_7(z)$. Since

$$\nu_2 \xleftarrow{\beta_1;\gamma_1;\eta_1;\eta_2} \nu_1 \xrightarrow{\beta_2;\gamma_2;\eta_1} \nu_7$$

   and z, being in $I(\alpha)$, is not an output variable of the involved expressions, we have $\nu_7(z) = \nu_2(z)$. Since $\nu_3 \xrightarrow{\eta_2} \nu_2$ with η_2 not touching z, we obtain $\nu_2(z) = \nu_3(z) = \nu_2(y)$.

(5) $O(\beta_1) - \rho_1(O(\alpha_1))$. This case is symmetrical to the previous one.

The above five cases confirm $\nu_8 = \nu_2$ which gives the contradiction, showing $(\nu_1, \nu) \notin [\![\alpha_2]\!]_D$ whence $(\nu_1, \nu) \in [\![\alpha]\!]_D$. In case (1) and case (3) we already observed that $\nu(y) = \nu_2(\rho_1(y))$ for $y \in O(\alpha_1) = O(\alpha)$. Thus, soundness is proved.

6.3. **From io-disjoint FLIF to Executable FO.** In this section we prove Theorem 5.4. Recall the translation given in Table 2. In order to prove Theorem 5.4, using Lemma 4.11, it suffices to prove the following:

Claim 6.1. For each α, the formula φ_α is $I(\alpha)$-executable and $\mathit{fvars}(\varphi_\alpha) = \mathit{vars}(\alpha)$. Moreover, for any instance D and any valuation ν, we have

$$(\nu, \nu) \in [\![\alpha]\!]_D \iff D, \nu \models \varphi.$$

Proof. By structural induction. The implication from left to right is referred to as *completeness*, and the other implication as *soundness*.

Atomic expressions. If α is $R(\bar{x};\bar{y})$, only soundness is not immediate. If $D,\nu \models \varphi$, then $\nu(\bar{x}) \cdot \nu(\bar{y}) \in D(R)$. Hence, $(\nu,\nu) \in [\![\alpha]\!]_D$, since two identical valuations agree trivially outside $O(\alpha)$. The cases where α is of the form $(x = y)$, $(x := y)$, $(x = c)$, or $(x := c)$, are immediate.

Next, we verify the inductive cases. In each step of the induction, we refer to φ_α simply as φ.

Composition. Consider α of the form $\alpha_1 ; \alpha_2$. We begin by checking that φ_α is $I(\alpha)$-executable. By Proposition 4.13, $I(\alpha_1)$ is disjoint from both $O(\alpha_1)$ and $O(\alpha_2)$.

Let $\varphi_1' = \exists_{O(\alpha_1)\cap O(\alpha_2)}\varphi_1$. Then $\mathit{fvars}(\varphi_1') = I(\alpha_1) \cup (O(\alpha_1) - O(\alpha_2))$. Indeed,

$$\begin{aligned}\mathit{fvars}(\varphi_1') &= \mathit{fvars}(\varphi_1) - (O(\alpha_1) \cap O(\alpha_2)) \\ &= \mathit{vars}(\alpha_1) - (O(\alpha_1) \cap O(\alpha_2)) \\ &= (I(\alpha_1) \cup O(\alpha_1)) - (O(\alpha_1) \cap O(\alpha_2)) \\ &= I(\alpha_1) \cup (O(\alpha_1) - O(\alpha_2))\end{aligned}$$

By induction, φ_1 is $I(\alpha_1)$-executable and φ_2 is $I(\alpha_2)$-executable. Let $\mathcal{V} = I(\alpha) = I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1))$. For φ to be $\mathcal{V}$-executable, it must be the case that:

- φ_1' is $\mathcal{V}$-executable, which means that φ_1 should be $\mathcal{V} - (O(\alpha_1) \cap O(\alpha_2))$-executable. Since $I(\alpha_1) \cap O(\alpha_1) = \emptyset$, we have $I(\alpha_1) \cap (O(\alpha_1) \cap O(\alpha_2)) = \emptyset$. This shows that $I(\alpha_1) \subseteq \mathcal{V} - (O(\alpha_1) \cap O(\alpha_2))$. Consequently, φ_1' is $\mathcal{V}$-executable.
- φ_2 is $\mathcal{V} \cup \mathit{fvars}(\varphi_1')$-executable, which means that φ_2 should be $\mathcal{V} \cup (I(\alpha_1) \cup (O(\alpha_1) - O(\alpha_2)))$-executable. We know that

$$\begin{aligned}\mathcal{V} \cup \mathit{fvars}(\varphi_1') &= \mathcal{V} \cup (I(\alpha_1) \cup (O(\alpha_1) - O(\alpha_2))) \\ &= I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1)) \cup (O(\alpha_1) - O(\alpha_2)) \\ &= I(\alpha_1) \cup ((I(\alpha_2) \cup O(\alpha_1)) - O(\alpha_2))\end{aligned}$$

  Since $I(\alpha_2) \cap O(\alpha_2) = \emptyset$, we have $I(\alpha_2) \subseteq \mathcal{V} \cup \mathit{fvars}(\varphi_1')$. Hence, φ_2 is $\mathcal{V} \cup \mathit{fvars}(\varphi_1')$-executable.

We next prove completeness. To this end, assume that $(\nu,\nu) \in [\![\alpha]\!]_D$. Then there exists a valuation ν' such that

(1) $(\nu,\nu') \in [\![\alpha_1]\!]_D$;
(2) $(\nu',\nu) \in [\![\alpha_2]\!]_D$.

Since $I(\alpha_2) \cap O(\alpha_2) = \emptyset$, Lemma 4.11 implies $(\nu,\nu) \in [\![\alpha_2]\!]_D$. Thus, by induction, $D,\nu \models \varphi_2$. Similarly from (1), we know that $D,\nu' \models \varphi_1$. Consequently,

$$D,\nu' \models \varphi_1' \tag{6.3}$$

Additionally, we know from (1) and (2) and the law of inertia $\nu = \nu'$ outside $O(\alpha_1)$ and outside $O(\alpha_2)$. Hence,

$$\nu = \nu' \text{ outside } O(\alpha_1) \cap O(\alpha_2) \tag{6.4}$$

From (6.3) and (6.4), we obtain $D,\nu \models \varphi_1'$, whence $D,\nu \models \varphi$.

To show soundness, assume $D,\nu \models \varphi$. Then

(1) $D,\nu \models \varphi_1'$, which means that there exists ν' such that
  (i) $\nu' = \nu$ outside $O(\alpha_1) \cap O(\alpha_2)$;
  (ii) $D,\nu' \models \varphi_1$.
(2) $D,\nu \models \varphi_2$.

By induction from (ii), we know that $(\nu', \nu') \in [\![\alpha_1]\!]_D$. Since $I(\alpha_1) \cap O(\alpha_1) = \emptyset$, we know from (i) that ν agrees with ν' on $I(\alpha_1)$ and outside $O(\alpha_1)$. Hence, we know by Lemma 4.6 that $(\nu, \nu') \in [\![\alpha_1]\!]_D$. Similarly, from (2), $(\nu', \nu) \in [\![\alpha_2]\!]_D$. Consequently, $(\nu, \nu) \in [\![\alpha]\!]_D$.

Difference. By induction, we know that φ_1 is $I(\alpha_1)$-executable and φ_2 is $I(\alpha_2)$-executable. Let $\mathcal{V} = I(\alpha) = I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$. By Proposition 4.13, we have $O(\alpha_1) \subseteq O(\alpha_2)$, so $\mathcal{V} = I(\alpha) = I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_2) - O(\alpha_1))$.

For φ to be $\mathcal{V}$-executable, we must verify the following:

- φ_1 is $\mathcal{V}$-executable and φ_2 is $\mathcal{V} \cup \mathit{fvars}(\varphi_1)$-executable. This holds since $I(\alpha_i) \subseteq \mathcal{V}$ for $i \in \{1, 2\}$.
- $\mathit{fvars}(\varphi_2) \subseteq \mathcal{V} \cup \mathit{fvars}(\varphi_1)$. We verify this as follows.

$$\begin{aligned} \mathit{fvars}(\varphi_2) &= I(\alpha_2) \cup O(\alpha_2) \\ &\subseteq I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_2) - O(\alpha_1)) \cup O(\alpha_1) \\ &= \mathcal{V} \cup I(\alpha_1) \cup O(\alpha_1) = \mathcal{V} \cup \mathit{fvars}(\varphi_1). \end{aligned}$$

Union. By induction, we know that φ_1 is $I(\alpha_1)$-executable and φ_2 is $I(\alpha_2)$-executable. Let $\mathcal{V} = I(\alpha) = I(\alpha_1) \cup I(\alpha_2) \cup (O(\alpha_1) \triangle O(\alpha_2))$. By Proposition 4.13 we have $O(\alpha_1) = O(\alpha_2)$, so $\mathcal{V} = I(\alpha_1) \cup I(\alpha_2)$.

For φ to be $\mathcal{V}$-executable, we must verify the following:

- φ_1 is $\mathcal{V}$-executable and φ_2 is $\mathcal{V}$-executable. This holds since $I(\alpha_i) \subseteq \mathcal{V}$ for $i \in \{1, 2\}$.
- $\mathit{fvars}(\varphi_1) \triangle \mathit{fvars}(\varphi_2) \subseteq \mathcal{V}$. We verify this as follows. Since $O(\alpha_1) = O(\alpha_2)$ and $I(\alpha_i) \cap O(\alpha_i) = \emptyset$ for $i = 1, 2$, we can reason as follows: (we use O to abbreviate $O(\alpha_1)$)

$$\begin{aligned} \mathit{fvars}(\varphi_1) \triangle \mathit{fvars}(\varphi_2) &= (I(\alpha_1) \cup O) \triangle (I(\alpha_2) \cup O) = I(\alpha_1) \triangle I(\alpha_2) \\ &\subseteq I(\alpha_1) \cup I(\alpha_2) = \mathcal{V}. \end{aligned}$$

□

7. Relational algebra plans for io-disjoint FLIF

In this section we show how the evaluation problem for FLIF$^{\text{io}}$ expressions can be solved in a very direct manner, using a translation into a particularly simple form of relational algebra plans.

We generalize the evaluation problem so that it can take a set of valuations as input, rather than just a single valuation. Formally, for an FLIF$^{\text{io}}$ expression α over a database schema $\mathcal{S}$, an instance D of $\mathcal{S}$, and a set N of valuations on $I(\alpha)$, we want to compute

$$\mathit{Eval}_\alpha(D, N) := \bigcup \{\mathit{Eval}_\alpha(D, \nu_{\text{in}}) \mid \nu_{\text{in}} \in N\}.$$

Viewing variables as attributes, we can view a set of valuations on a finite set of variables Z, like the set N above, as a relation with relation schema Z. Consequently, it is convenient to use the named perspective of the relational algebra [AHV95], where every expression has an output relation schema (a finite set of attributes; variables in our case). We briefly review the well-known operators of the relational algebra and their behavior on the relation schema level:

- Union and difference are allowed only on relations with the same relation schema.
- Natural join ($\bowtie$) can be applied on two relations with relation schemas Z_1 and Z_2, and produces a relation with relation schema $Z_1 \cup Z_2$.

- Projection (π) produces a relation with a relation schema that is a subset of the input relation schema.
- Selection (σ) does not change the schema.
- Renaming will not be needed. Instead, however, to accommodate the assignment expressions present in FLIF$^{\text{io}}$, we will need the generalized projection operator that adds a new attribute with the same value as an existing attribute, or a constant. Let N be a relation with relation schema Z, let $y \in Z$, and let x be a variable not in Z. Then

$$\pi_{Z,x:=y}(N) = \{\nu[x := \nu(y)] \mid \nu \in N\}$$
$$\pi_{Z,x:=c}(N) = \{\nu[x := c] \mid \nu \in N\}$$

Plans are based on *access methods*, which have the following syntax and semantics. Let $R(\bar{x};\bar{y})$ be an atomic FLIF$^{\text{io}}$-expression. Let X be the set of variables in $\bar{x}$ and let Y be the set of variables in $\bar{y}$ (in particular, X and Y are disjoint). Let N be a relation with a relation schema Z that contains X but is disjoint from Y. Let D be a database instance. We define the result of the *access join* of N with $R(\bar{x};\bar{y})$, evaluated on D, to be the following relation with relation schema $Z \cup Y$:

$$N \overset{\text{access}}{\bowtie} R(\bar{x};\bar{y}) := \{\nu \text{ valuation on } Z \cup Y \mid \nu|_Z \in N \text{ and } \nu(\bar{x}) \cdot \nu(\bar{y}) \in D(R)\}$$

This result relation can clearly be computed respecting the limited access pattern on R. Indeed, we iterate through the valuations in N, feed their X-values to the source R, and extend the valuations with the obtained Y-values.

Formally, over any database schema $\mathcal{S}$ and for any finite set of variables I, we define a *plan over $\mathcal{S}$ with input variables I* as an expression that can be built up as follows:

- The special relation name In, with relation schema I, is a plan.
- If $R(\bar{x};\bar{y})$ is an atomic FLIF$^{\text{io}}$ expression over $\mathcal{S}$, with sets of variables X and Y as above, and E is a plan with output relation schema Z as above, then also $E \overset{\text{access}}{\bowtie} R(\bar{x};\bar{y})$ is a plan, with output relation schema $Z \cup Y$.
- Plans are closed under union, difference, natural join, and projection.

Given a database instance D, a set N of valuations on I, and a plan E with input variables I, we can instantiate the relation name In by N and evaluate E on (D, N) in the obvious manner. We denote the result by $E(D, N)$.

We establish:

Theorem 7.1. *For every* FLIF$^{\text{io}}$ *expression α over a database schema $\mathcal{S}$ there exists a plan E_α over $\mathcal{S}$ with input variables $I(\alpha)$, such that $Eval_\alpha(D, N) = E_\alpha(D, N)$, for every instance D of $\mathcal{S}$ and set N of valuations on $I(\alpha)$.*

Example 7.2.

- Let α be $R(x;y)\,;S(y;z)$. Recall that $I(\alpha) = \{x\}$. A plan for α can be taken to be
$$(In \overset{\text{access}}{\bowtie} R(x;y)) \overset{\text{access}}{\bowtie} S(y;z).$$
- Let α be $R(x_1;y,u)\,;S(x_2,y;z,u)$. Recall that $I(\alpha) = \{x_1, x_2\}$. A plan for α can be taken to be
$$\pi_{x_1,x_2,y}(In \overset{\text{access}}{\bowtie} R(x_1;y,u)) \overset{\text{access}}{\bowtie} S(x_2,y;z,u).$$
- Recall the expression $R(x;y_1) \cup S(x;y_2)$ from Example 4.1, which has input variables $\{x, y_1, y_2\}$ and no output variables. A plan for this expression is
$$(\pi_{x,y_2}(In) \overset{\text{access}}{\bowtie} R(x;y_1)) \bowtie In \;\cup\; (\pi_{x,y_1}(In) \overset{\text{access}}{\bowtie} S(x;y_2)) \bowtie In.$$

The joins with In ensure that the produced output values are equal to the given input values, which may be needed in case N has multiple tuples.

Proof. To prove the theorem we need a stronger induction hypothesis, where we allow N to have a larger relation schema $Z \supseteq I(\alpha)$, while still being disjoint with $O(\alpha)$. The claim then is that

$$E_\alpha(D, N) = \{\nu \text{ on } Z \cup O(\alpha) \mid \nu|_{vars(\alpha)} \in Eval_\alpha(D, \nu|_{I(\alpha)})\}.$$

The base cases are clear. If α is $R(\bar{x};\bar{y})$, then E_α is $In \overset{\text{access}}{\bowtie} R(\bar{x};\bar{y})$. If α is $(x = y)$, then E_α is the selection $\sigma_{x=y}(In)$. If α is $(x := y)$, then E_α is the generalized projection $\pi_{y,x:=y}(In)$.

In what follows we use the following notation. Let P and Q be plans. By $Q(P)$ we mean the plan obtained from Q by substituting P for In.

Suppose α is $\alpha_1\,;\alpha_2$. Plan E_{α_1}, obtained by induction, assumes an input relation schema that contains $I(\alpha_1)$ and is disjoint from $O(\alpha_1)$. Since $I(\alpha) = I(\alpha_1) \cup (I(\alpha_2) - O(\alpha_1))$, $I(\alpha_1) \cap O(\alpha_1) = \emptyset$, and Z is disjoint from $O(\alpha) = O(\alpha_1) \cup O(\alpha_2)$, we can apply E_{α_1} with input relation schema Z. Let P_1 be the plan $\pi_{Z-O(\alpha_2)}(E_{\alpha_1})$. Then E_α is the plan $E_{\alpha_2}(P_1)$. (One can again verify that this is a legal plan.)

Next, suppose α is $\alpha_1 \cup \alpha_2$. Then $I(\alpha) = I(\alpha_1) \cup I(\alpha_2)$, which is disjoint from $O(\alpha_1) = O(\alpha_2)$ (compare Proposition 4.13). Hence, for E_α we can simply take the plan $E_{\alpha_1} \cup E_{\alpha_2}$.

Finally, suppose α is $\alpha_1 - \alpha_2$. Then E_α is

$$E_{\alpha_1} - (E_{\alpha_2}(\pi_{I(\alpha)-O(\alpha_2)}(In)) \bowtie In.$$

In general, in the above translations, we follow the principle that the result of a subplan E_{α_i} must be joined with In whenever $O(\alpha_i)$ may intersect with $I(\alpha)$. □

Remark 7.3. When we extend plans with assignment statements such that common expressions can be given a name [BLtCT16], the translation given in the above proof leads to a plan E_α of size linear of the length of α. Each time we do a substitution of a subexpression for In in the proof, we first assign a name to the subexpression and only substitute the name.

Example 7.4. Recall the query from Example 2.5 expressed in FLIF$^{\text{io}}$ slightly differently as follows:

$$F(x; y_1)\,;F(x; y_2)\,;(F(y_1; z) \cap F(y_2; z))\,;(y_1 \neq y_2)$$

The plan equivalent to this expression is:

$$\sigma_{y_1 \neq y_2}((E \bowtie F(y_1; z)) \bowtie (E \bowtie F(y_2; z)))$$

where E is the partial plan $In \bowtie F(x; y_1) \bowtie F(x; y_2)$ with In a relation name over $\{x\}$ providing input values.

8. Related Work

Much of the work on the topic of information sources with access limitations has been on processing queries expressed in generic query languages, such as conjunctive queries, unions of conjunctive queries, conjunctive queries with negation, first-order logic (relational calculus), or Datalog. Here, the query is written as if the database has no access limitations; the challenge then is to find a query plan that does respect the limitations, but produces,

ideally, the same answers, or, failing that, produces only correct answers (also known as sound rewritings) [DGL00].

Query plans could take the form of syntactically ordered fragments of the query languages that are used, like executable FO considered in the present paper [RSU95, Li03, NL04]. Query plans can also be directly described in relational algebra, like the plans defined here in Section 7 [YLGMU99, FLMS99, BtCT16]. An alternative approach to query processing under access limitations is to first retrieve the "accessible part" of the database; after that we can simply execute the original query on that part, which is a sound strategy for monotone queries. Computing the accessible part may require recursion; on the other hand, the computation can be optimized so as to contain only information needed for the specific query [CM08b].

When the query language used is first-order logic, the planning and optimization problems mentioned above are, of course, undecidable. Yet, a remarkable preservation theorem [BLtCT16] states that, assuming a given first-order query only depends on the accessible part of the database (for any database; this is a semantic and undecidable property), then, that query can actually be rewritten into an executable FO formula.

Interestingly, a variant of our translation results from FLIF to executable FO in Proposition 3.7 can be seen to follow from the preservation theorem just mentioned. It would suffice to express a given FLIF expression α by any first-order logic formula φ_α in the free variables $\mathbb{V}_x \cup \mathbb{V}_y$, without taking care that φ_α is executable. Indeed, FLIF expressions are readily seen to be access-determined by the variables in $\mathbb{V}_x$, so, the preservation theorem would imply that φ_α can be equivalently written by an executable formula. Of course, our result provides a much more direct translation, and moreover, shows a bound on the number of variables (free or bound) needed in φ_α.

Furthermore, it is natural to expect (although we have not verified it formally) that any FLIF expression α is already access-determined by the set of its input variables $I(\alpha)$. In this manner, also Theorem 5.4 would be implied by the preservation theorem. Again, our theorem provides a direct and actually very efficient translation.

Incidentally, in the cited work [BLtCT16], Benedikt et al. define their own notion of executable FO, syntactically rather different from the one we use in the present paper (which was introduced much earlier by Nash and Ludäscher [NL04]). We prefer the language we use for its elegance, and because its treatment of input variables matches well with input variables for FLIF expressions. Still, both executable-FO languages are equivalent in expressive power, as they are both equivalent to the plans used here in Section 7 and also used by Benedikt et al.

In a companion paper [ABS$^+$23], first presented at the KR 2020 conference, we consider LIF in a broader (but still first-order) context, independently of access limitations. The companion paper considers the problem of sensitivity analysis for general LIF expressions and introduces semantic as well as syntactic definitions of input and output variables. the syntactic definitions were shown to be optimal approximations of the semantic definitions. We have adopted the syntactic treatment in this paper, and have shown its relevance, when applied it to FLIF, to querying information sources with access limitations. Propositions 4.3, 4.4 and 4.5 are adopted here from our companion paper (there, numbered Proposition 4.3, Lemma 4.4, and Lemma 4.6, respectively); the proofs can be found there.

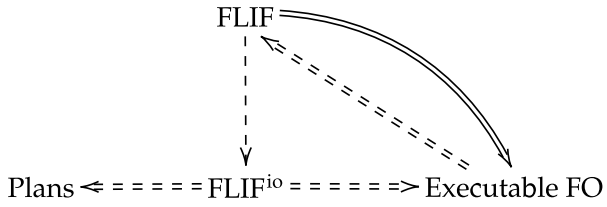

Figure 3: Summary of the translations shown in the paper, where a dashed arrow denotes an input-respecting translation, a double arrow denotes a simple translation, and finally, a double dashed arrow denotes both.

9. Conclusion

We have presented a connection between executable queries on databases with access limitations on the one hand, and first-order dynamic logic frameworks on the other hand. Specifically, we have defined Forward LIF (FLIF), an instantiation of the Logic of Information Flows (LIF). FLIF presents itself as an XPath-like language for graphs of valuations, where edges represent information accesses. The main novelty of FLIF lies in its graph-navigational nature (without explicit quantification), its input-output mechanism, and the law of inertia that it obeys. Specifically for io-disjoint FLIF expressions, our work also presents a more transparent alternative to the result by Benedikt et al. on translating (their version of) executable first-order formulas to plans. We have also given renewed attention to Nash and Ludäscher's elegant executable FO language, which seemed to have been overlooked by more recent research in the field.

Figure 3 illustrates our main technical results, which offer translations between various languages. Most of the translations are simple in their formulation, although the rigorous proof of correctness is not always that simple.

We are not claiming that FLIF is necessarily more user-friendly than previous languages, or necessarily easier to implement or optimize. Both of these aspects should be the topic of further research. Still we believe it offers a novel perspective. That FLIF can express all executable FO queries is something that is not obvious at first sight.

In closing, we note that querying under limited access patterns has applicability beyond traditional data or information sources. For instance in the context of distributed data, when performing tasks involving the composition of external services, functions, or modules, limited access patterns are a way for service providers to protect parts of their data, while still allowing their services to be integrated seamlessly in other applications. Limited access patterns also have applications in active databases, where we like to think of FLIF as an analog of Active XML [ABM08] for the relational data model.

Acknowledgment

This research was partially supported by the Flanders AI Research Program. We thank the anonymous reviewers for their critical comments on an earlier version of this paper, which prompted us to significantly improve the presentation of our results.

References

[AAB+17] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017.

[ABM08] S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *The VLDB Journal*, 17(5):1019–1040, 2008.

[ABS+20] H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska, and J. Van den Bussche. Executable first-order queries in the logic of information flows. In *Proceedings 23rd International Conference on Database Theory*, volume 155 of *Leibniz International Proceedings in Informatics*, pages 4:1–4:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

[ABS+23] H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska, and J. Van den Bussche. Inputs, outputs, and composition in the logic of information flows. *ACM Transactions on Computational Logic*, 24(4):33:1–33:44, 2023.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AVdB21] H. Aamer and J. Van den Bussche. Input-Output Disjointness for Forward Expressions in the Logic of Information Flows. In Ke Yi and Zhewei Wei, editors, *24th International Conference on Database Theory (ICDT 2021)*, volume 186 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[BBB13] V. Bárány, M. Benedikt, and P. Bourhis. Access patterns and integrity constraints revisited. In W.-C. Tan et al., editors, *Proceedings 16th International Conference on Database Theory*, pages 213–224. ACM, 2013.

[BGS11] M. Benedikt, G. Gottlob, and P. Senellart. Determining relevance of accesses at runtime. In *Proceedings 30st ACM Symposium on Principles of Databases*, pages 211–222. ACM, 2011.

[BLT15] M. Benedikt, J. Leblay, and E. Tsamoura. Querying with access patterns and integrity constraints. *Proceedings of the VLDB Endownment*, 8(6):690–701, 2015.

[BLtCT16] M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Morgan & Claypool, 2016.

[BtCT16] M. Benedikt, B. ten Cate, and E. Tsamoura. Generating plans from proofs. *ACM Transactions on Database Systems*, 40(4):22:1–22:45, 2016.

[CCM09] A. Calı, D. Calvanese, and D. Martinenghi. Dynamic query optimization under access limitations and dependencies. *Journal of Universal Computer Science*, 15(1):33–62, 2009.

[CM08a] A. Calı and D. Martinenghi. Conjunctive query containment under access limitations. In Q. Li, S. Spaccapietra, et al., editors, *Proceedings 27th International Conference on Conceptual Modeling*, volume 5231 of *Lecture Notes in Computer Science*, pages 326–340. Springer, 2008.

[CM08b] A. Calı and D. Martinenghi. Querying data under access limitations. In *Proceedings 24th International Conference on Data Engineering*, pages 50–59. IEEE Computer Society, 2008.

[CMRU17] A. Calı, D. Martinenghi, I. Razon, and M. Ugarte. Querying the deep web: Back to the foundations. In Reutter and Srivastava [RS17].

[CU18] A. Calı and M. Ugarte. On the complexity of query answering under access limitations: A computational formalism. In D. Olteanu and B. Poblete, editors, *Proceedings 12th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 2100 of *CEUR Workshop Proceedings*, 2018.

[DGL00] O.M. Duschka, M.R. Genesereth, and A.Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–73, 2000.

[DLN07] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *Theoretical Computer Science*, 371(3):200–226, 2007.

[FGL+15] G.H.L. Fletcher, M. Gyssens, D. Leinders, D. Surinx, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. *Information Sciences*, 298:390–406, 2015.

[FLMS99] D. Florescu, A.Y. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In SIGMOD99 [SIG99], pages 311–322.

[GS91] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[IL84] T. Imielinski and W. Lipski. The relational model of data and cylindric algebras. *Journal of Computer and System Sciences*, 28:80–102, 1984.

[Li03] C. Li. Computing complete answers to queries in the presence of limited access patterns. *The VLDB Journal*, 12(3):211–227, 2003.

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[LMV13] L. Libkin, W. Martens, and D. Vrgoč. Quering graph databases with XPath. In *Proceedings 16th International Conference on Database Theory*. ACM, 2013.

[Mad91] R.D. Maddux. The origin of relation algebras in the development and axiomatization of the calculus of relations. *Studia Logica*, 50(3/4):421–455, 1991.

[MHF03] T.D. Millstein, A.Y. Halevy, and M.T. Friedman. Query containment for data integration systems. *Journal of Computer and System Sciences*, 66(1):20–39, 2003.

[NL04] A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *Proceedings 23th ACM Symposium on Principles of Database Systems*, pages 307–318, 2004.

[PAG10] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

[Pra92] V. Pratt. Origins of the calculus of binary relations. In *Proceedings 7th Annual IEEE Symposium on Logic in Computer Science*, pages 248–254, 1992.

[RS17] J.L. Reutter and D. Srivastava, editors. *Proceedings 11th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1912 of *CEUR Workshop Proceedings*, 2017.

[RSU95] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering queries using templates with binding patterns. In *Proceedings Fourteenth ACM Symposium on Principles of Database Systems*, pages 105–112. ACM Press, 1995.

[SFG+15] D. Surinx, G.H.L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs using transitive closure. *Logic Journal of the IGPL*, 23(5):759–788, 2015.

[SIG99] *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, volume 28:2 of *SIGMOD Record*. ACM Press, 1999.

[Tar41] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[tCM07] B. ten Cate and M. Marx. Navigational XPath: Calculus and algebra. *SIGMOD Record*, 36(2):19–26, 2007.

[Ter17] E. Ternovska. Recent progress on the algebra of modular systems. In Reutter and Srivastava [RS17].

[Ter19] E. Ternovska. An algebra of modular systems: static and dynamic perspectives. In A. Herzig and A. Popescu, editors, *Frontiers of Combining Systems: Proceedings 12th FroCos*, volume 11715 of *Lecture Notes in Artificial Intelligence*, pages 94–111. Springer, 2019.

[VdB01] J. Van den Bussche. Applications of Alfred Tarski's ideas in database theory. In L. Fribourg, editor, *Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*. Springer, 2001.

[YLGMU99] R. Yerneni, C. Li, H. Garcia-Molina, and J.D. Ullman. Computing capabilities of mediators. In SIGMOD99 [SIG99], pages 443–454.