



Input–Output Disjointness for Forward Expressions in the Logic of Information Flows

Heba Aamer  

Hasselt University, Belgium

Jan Van den Bussche  

Hasselt University, Belgium

Abstract

Last year we introduced the logic FLIF (forward logic of information flows) as a declarative language for specifying complex compositions of information sources with limited access patterns. The key insight of this approach is to view a system of information sources as a graph, where the nodes are valuations of variables, so that accesses to information sources can be modeled as edges in the graph. This allows the use of XPath-like navigational graph query languages. Indeed, a well-behaved fragment of FLIF, called io-disjoint FLIF, was shown to be equivalent to the executable fragment of first-order logic. It remained open, however, how io-disjoint FLIF compares to general FLIF. In this paper we close this gap by showing that general FLIF expressions can always be put into io-disjoint form.

2012 ACM Subject Classification Software and its engineering → Semantics; Software and its engineering → Data flow languages; Theory of computation → Database query languages (principles)

Keywords and phrases Composition, expressive power, variable substitution

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.8

Funding This work was partially supported by Artificial Intelligence Research Flanders.

Heba Aamer: Supported by the Special Research Fund (BOF) (BOF19OWB16).

Jan Van den Bussche: Partially supported by the National Natural Science Foundations of China (61972455).

Acknowledgements Thanks to Eugenia Ternovska for introducing us to LIF and to Bart Bogaerts for initial discussions on the topic of this paper.

1 Introduction

FLIF (Forward Logic of Information Flows) [1] is an algebraic language for accessing atomic modules, and composing such modules into complex transition systems. Here, the term “atomic module” can be interpreted liberally: it can be a software library function, a Web service, a form on a website, or an information source.

Abstractly, an atomic module may be viewed as an n -ary relation where some of the arguments are designated as input arguments, with the remaining arguments providing output. For a simple example, a telephone directory may be viewed as a binary relation $\text{Dir}(\text{name}; \text{phone})$ with name as input argument and phone as output argument. For another example, the public bus company may provide its weekdays schedule as a relation $\text{Route}(\text{stop}, \text{interval}; \text{time}, \text{line}, \text{next}, \text{duration})$ that, given a 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 input arguments from the output arguments.

In database research, such relations are known as information sources with limited access patterns, and the querying of such sources has received considerable attention. We refer to the research monograph by Benedikt et al. for more background [7]. An elegant syntactic fragment of first-order logic (FO) that takes into account the limitations imposed by the access patterns, was proposed by Nash and Ludäscher in the form of so-called “executable”



© Heba Aamer and Jan Van den Bussche;
licensed under Creative Commons License CC-BY 4.0
24th International Conference on Database Theory (ICDT 2021).

Editors: Ke Yi and Zhewei Wei; Article No. 8; pp. 8:1–8:18

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

FO formulas [14]. Executable FO queries can be evaluated by a form of relational algebra expressions, called *plans*, in which 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.

FLIF now offers an alternative language, which we think of as situated halfway between executable FO and plans. In FLIF we take a novel graph-based perspective to information sources with access patterns. The nodes of the graph are variable bindings; edges indicate accesses to the sources. For example, consider a source `Friend(pname; fname)` that outputs names of friends when given the name of a person as input. Then there is an edge labeled `Friend` from binding ν_1 to binding ν_2 if $\nu_2(\text{fname})$ is a friend of $\nu_1(\text{pname})$. Moreover, ν_2 should not differ from ν_1 in other variables (a principle of “inertia”). FLIF then is a simple XPath-like navigational query language over such graphs [15, 10, 5, 9, 16, 4].

For example, abbreviating `Friend` by F , the FLIF expression $F(x; y); F(y; z); F(z; u); (u = x)$ retrieves (in variable z) friends of friends of x who have x also as a direct friend. Here, the operator `;` denotes composition and the subexpression $(u = x)$ serves as a test. Composition is a crucial operator by which (bounded-length) paths can be traced in the graph. FLIF also has union (to branch off), intersection (to merge branches) and difference (to exclude branches), and variable assignments.

Simple as FLIF may seem, it is at least as expressive as executable FO, as we showed together with Bogaerts, Surinx and Ternovska at ICDT 2020 [1]. Actually, executable FO was shown to be already subsumed by a well-behaved fragment of FLIF, formed by the expressions that are *io-disjoint*. IO-disjointness is a syntactic restriction which guarantees that, during the transitions involved in evaluating the expression, the values of input variables are never overwritten. IO-disjoint expressions can also be evaluated by a very transparent translation to relational algebra plans, in which all joins are natural joins, and attribute renaming is not needed. (The example expression above is io-disjoint.)

Conversely, in our previous work we also gave a translation from io-disjoint FLIF to executable FO. It has remained open, however, whether every FLIF expression can actually be put in an equivalent form that is io-disjoint. In the present paper we answer this question affirmatively. Of course, we need to make precise for what kind of “equivalence” this can work, and it is part of our contribution to clarify this. Intuitively, we show that it is always possible to designate a fresh set of output variables disjoint from the set of input variables, in such a way that intermediate variables (used in subexpressions) do not interfere with either of the two sets. Proving this rigorously turned out to be a quite intricate task. Our result shows that io-disjoint FLIF is equally powerful as the full FLIF language. As a corollary, we obtain that full FLIF is not more powerful than executable FO.

This paper is organized as follows. Section 2 gives preliminaries. Section 3 gives examples of queries expressed in FLIF and other languages. Section 4 presents and discusses our main result. Section 5 formally proves the correctness of our method. Section 6 concludes.

2 Preliminaries

We begin by recalling from previous work the concepts and results needed for the present paper.

2.1 Syntax of FLIF

A *schema* consists of a set of *module names*, which are seen as relation names. Accordingly, the schema assigns to each module name M an *arity* $\text{ar}(M)$, as well as an *input arity* $\text{iar}(M)$ with $\text{iar}(M) \leq \text{ar}(M)$. The idea is that the first $\text{iar}(M)$ arguments are the input arguments;

the remaining arguments are the output arguments. We use $\text{oar}(M)$ (output arity) for $\text{ar}(M) - \text{iar}(M)$.

Assume countably infinite universes **dom** of data values, and \mathbb{V} of variables. The expressions α of FLIF over a schema \mathcal{S} are generated by the following grammar:

$$\alpha ::= M(\bar{x}; \bar{y}) \mid (x := y) \mid (x := c) \mid (x = y) \mid (x = c) \mid \alpha ; \alpha \mid \alpha \cup \alpha \mid \alpha \cap \alpha \mid \alpha - \alpha$$

Here, M is a module name from \mathcal{S} ; \bar{x} and \bar{y} are tuples of variables of lengths $\text{iar}(M)$ and $\text{oar}(M)$ respectively; x and y are variables; and c is a constant from **dom**. An expression of the form $M(\bar{x}; \bar{y})$ is called a *module access*; of the form $x := y$ or $x := c$ a *variable assignment*; and $x = y$ or $x = c$ an *equality test*.

► **Remark 1.** In writing 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.

2.2 Semantics of FLIF

Recall that a *valuation* is a mapping from \mathbb{V} to **dom**. It is convenient to be able to apply valuations also to constants, agreeing that $\nu(c) = c$ for any valuation ν and any $c \in \mathbf{dom}$. We use \mathcal{V} for the set of all valuations.

The semantics of FLIF expressions are defined in the context of interpretations. An *interpretation* D of a schema \mathcal{S} assigns to each module name M an n -ary relation $D(M)$ on **dom**, with $n = \text{ar}(M)$. Given an interpretation D , we define the semantics of an expression α on D , denoted by $\llbracket \alpha \rrbracket_D$, as a binary relation on \mathcal{V} , as follows.

For module access:

$$\llbracket M(\bar{x}; \bar{y}) \rrbracket_D = \{(\nu_1, \nu_2) \in \mathcal{V} \times \mathcal{V} \mid \nu_1(\bar{x}) \cdot \nu_2(\bar{y}) \in D(M) \text{ and } \nu_2 \text{ agrees with } \nu_1 \text{ outside } \bar{y}\}$$

where \cdot denotes concatenation.

For variable assignment, where t is a variable or a constant:

$$\llbracket x := t \rrbracket_D = \{(\nu_1, \nu_2) \in \mathcal{V} \times \mathcal{V} \mid \nu_2 = \nu_1[x := \nu_1(t)]\}$$

where, in general, for a valuation ν , a variable x , and a data value c , we use $\nu[x := c]$ for the valuation that is the same as ν except that x is mapped to c .

For equality test:

$$\llbracket x = t \rrbracket_D = \{(\nu, \nu) \mid \nu \in \mathcal{V} \text{ and } \nu(x) = \nu(t)\}.$$

Finally, the semantics of the operators $;$, \cup , \cap and $-$ are given by composition of binary relations, union, intersection and set difference, respectively.¹

► **Remark 2.** For simplicity of exposition, we only include equality tests and simple assignments. The definitions and results of this paper can however be extended to include arithmetical comparisons and operations.

► **Example 3.** Suppose F is a binary relation of input arity one that is interpreted as a symmetric “friends” relation of a social network. Suppose the value of input variable x is some famous person, and we want to find persons z who are friend of a friend (say, y) of x .

¹ Recall that the composition $r ; s$ of two binary relations equals the binary relation $\{(\nu_1, \nu_3) \mid \exists \nu_2 : (\nu_1, \nu_2) \in r \text{ and } (\nu_2, \nu_3) \in s\}$.

8:4 Input–Output Disjointness for FLIF

■ **Table 1** Input and output variables of FLIF expressions [1]. In the case of $M(\bar{x}; \bar{y})$, the set X is the set of variables in \bar{x} , and the set Y is the set of variables in \bar{y} . The symbol Δ denotes symmetric difference.

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

For this we can use the simple expression $F(x; y) ; F(y; z)$. In order to find persons z with a pair (say, y_1 and y_2) of friends of x , we can write the expression

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

If we want to make sure that y_1 and y_2 are different, we can write $\alpha - (\alpha ; (y_1 = y_2))$ where α is the previous expression. In view of Remark 2 above, we could easily extend FLIF with nonequalities and then write $\alpha ; (y_1 \neq y_2)$ instead.

2.3 Input and output variables

Intuitively, an *output variable* of an expression α is any variable whose value may change in the course of evaluating α . The set of *input variables* may then be defined as the smallest set of variables whose values determine the values of the output variables. These semantic notions of input and output are undecidable, as they are related to satisfiability in the algebra of binary relations [3]. Hence, we use syntactical overapproximations [1]. Thus Table 1 defines, for each expression α , the sets $I(\alpha)$ and $O(\alpha)$ of input and output variables.

The following inertia [12, 11] and input determinacy properties formally confirm that Table 1 defines correct overapproximations, i.e., $I(\alpha)$ contains all semantic input variables, and $O(\alpha)$ contains all semantic output variables. Elsewhere [2] we have shown that slight variants of our definitions of I and O are optimal among all compositional definitions satisfying the below two properties.

► **Proposition 4** ([1]). *Let α be an FLIF expression and let D be an interpretation.*

Inertia: *For any $(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D$, we have that ν_2 agrees with ν_1 outside $O(\alpha)$.*

Input determinacy: *Let $(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D$ 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 $(\nu'_1, \nu'_2) \in \llbracket \alpha \rrbracket_D$.

One can verify that every variable occurring in an expression according to the definitions in Table 1 is either an input variable, an output variable, or both. Since FLIF lacks an explicit quantification operator, we also refer to the variables occurring in an expression α as the “free variables”, denoted by $\text{var}(\alpha)$, so $\text{var}(\alpha) = I(\alpha) \cup O(\alpha)$. The following property

formalizes, as expected, that the evaluation of an expression is oblivious to the values of the non-free variables; they can take any values. (Of course by inertia, these values will not change in the course of evaluating the expression.)

► **Proposition 5** (Free variable property [1]). *Let Y be the set of variables not in $\text{var}(\alpha)$, let $(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D$, and let $\nu : Y \rightarrow \mathbf{dom}$ be arbitrary. Then also $(\nu_1[\nu], \nu_2[\nu]) \in \llbracket \alpha \rrbracket_D$.*

Here, we use $\nu_i[\nu]$ for ν_i updated with ν , or formally, the valuation $\nu_i|_{\text{var}(\alpha)} \cup \nu$.

► **Example 6.** 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 interpretation 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 $(\nu_1, \nu_2) \in \llbracket S(x; z) \rrbracket_D \subseteq \llbracket \alpha \rrbracket_D$. 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 $\llbracket S(x; z) \rrbracket_D$ nor to $\llbracket R(x; y) \rrbracket_D$ (due to inertia). Thus, input determinacy would be violated if y would not belong to $I(\alpha)$.

We mentioned that Table 1 overapproximates the undecidable semantic notions of inputs and outputs. The following example provides a simple illustration.

► **Example 7.** Let α be the expression $(x = y) ; R(x; y) ; (x = y)$. It is clear that α does not have any outputs, however $O(\alpha) = \{y\}$ according to the definitions of Table 1.

2.4 Input–output disjointness

An expression α is called *io-disjoint* if $I(\beta)$ and $O(\beta)$ are disjoint, for every subexpression β of α (including α itself). While there is nothing really “wrong” with expressions that are not io-disjoint, the io-disjoint expressions enjoy a particularly transparent evaluation process in which input slots remain intact while output slots are being filled.

► **Example 8.** Continuing Example 3 (friends), the expression $F(x; x)$ is obviously not io-disjoint. Evaluating this expression will overwrite variable x with a friend of the person originally stored in x . In contrast, the example expressions given in Example 3 are all io-disjoint. ◀

Formally, we have the following useful property, which follows from Proposition 4:

► **Proposition 9** (Identity property). *Let α be an io-disjoint expression and let D be an interpretation. If $(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D$, then also $(\nu_2, \nu_2) \in \llbracket \alpha \rrbracket_D$.*

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.

► **Example 10.** 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.

3 FLIF and other languages

In this Section we compare and contrast FLIF with other formalisms considered for querying over limited access patterns, specifically, executable FO; plans; and executable Datalog. For executable FO and plans we use the syntax introduced in our ICDT 2020 paper [1]. The syntax for Datalog needs no introduction.

8:6 Input–Output Disjointness for FLIF

► **Example 11.** Recall the Friends relation from Example 3. We will consider the query from that example in Example 12; here, we first consider a simpler query. Given a person x , we want in variable z the friends of a friend of x that are not friend with x itself.

In FLIF, we can express this query using only the variables x and z with the following simple expression:

$$F(x; z) ; F(z; z) - F(x; z)$$

The same query can be expressed with the following io-disjoint FLIF expression:

$$F(x; y) ; (F(y; z) - F(x; z))$$

The algebraic plan equivalent to this expression is:

$$\pi_z((In \bowtie F(x; y) \bowtie F(y; z)) - (In \bowtie F(x; y) \bowtie F(x; z)))$$

where In is a relation name over $\{x\}$ providing input values.

In executable FO, the query will be expressed by the formula:

$$\exists y F(x; y) \wedge F(y; z) \wedge \neg F(x; z)$$

Very similarly, in Datalog, the query can be expressed with the following program:

$$Q(z) \leftarrow F(x; y), F(y; z), \neg F(x; z).$$

► **Example 12.** Recall the query from Example 3 expressed in FLIF slightly differently as follows:

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

This expression is already io-disjoint. The plan equivalent to this expression is:

$$\pi_z \sigma_{y_1 \neq y_2}((In \bowtie F(x; y_1) \bowtie F(x; y_2) \bowtie F(y_1; z)) \cap (In \bowtie F(x; y_1) \bowtie F(x; y_2) \bowtie F(y_2; z)))$$

where In is a relation name over $\{x\}$ providing input values.

In executable FO, the query will be expressed by the formula:

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

In Datalog, the program could be the following:

$$A(y_1, y_2, z) \leftarrow F(x; y_1), F(x; y_2), F(y_1; z), F(y_2; z).$$

$$B(y_1, y_2) \leftarrow A(y_1, y_2, z), y_1 = y_2.$$

$$Q(z) \leftarrow A(y_1, y_2, z), \neg B(y_1, y_2).$$

4 Putting FLIF expressions in io-disjoint form

The main result of this paper is that arbitrary FLIF expressions can be put in io-disjoint form. 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.

4.1 Examples

An obvious approach to obtaining an io-disjoint expression is to rename 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.

When applying this 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 8, 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

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

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.

Expressions involving intersection can be rewritten with the aid of composition and equality test. For example the expression $R(x; y) \cap S(y; y)$ becomes

$$R(x; y_1); S(y; y_2); (y_1 = y_2).$$

The overall output y is renamed to y_1 , but in the rewriting of the subexpression $S(y; y)$ we use an intermediate output variable y_2 , then test that the two outputs are the same as required by the original expression. Note that this usage of intermediate variables is different from that used in the treatment of composition expressions. There, the intermediate variable was on the output side of the first subexpression and on the input side of the second subexpression; here, it is on the output side of the second subexpression.

An additional complication with intersection is that outputs that are not common to the lhs and rhs subexpressions of the intersection lose their status of output variable (Table 1), so must remain inertial for the overall expression. Hence, for the rewriting to have the desired semantics, we must add the appropriate equality tests. For example, the expression

$R(x, y; x, y) \cap S(x, z; x, z)$ has only x as an output, and x, y and z as inputs. Renaming the output x to x_1 , it can be rewritten in io-disjoint form as

$$R(x, y; x_1, y_1); (y_1 = y); S(x, z; x_2, z_1); (z_1 = z); (x_2 = x_1).$$

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

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

4.2 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:²

► **Theorem 13.** *Let α be an FLIF expression and let ρ be a bijection from $O(\alpha)$ to a set of variables disjoint from $\text{var}(\alpha)$. There exists an io-disjoint FLIF expression β such that*

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

$$\{\nu_2|_{O(\alpha)} \mid (\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D\} = \{\nu_2 \circ \rho \mid (\nu_1, \nu_2) \in \llbracket \beta \rrbracket_D\}.$$

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 a module 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 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 interpretation for which $\llbracket \alpha \rrbracket_D$ is nonempty. Hence $\llbracket \beta \rrbracket_D$ is nonempty as well. Now let D' be the interpretation with $D'(S) = D(S)$ but $D'(R) = \emptyset$. Then $\llbracket \alpha \rrbracket_{D'}$ becomes clearly empty, but $\llbracket \beta \rrbracket_{D'} = \llbracket \beta \rrbracket_D$ remains nonempty since β does not mention R .

² We use $g \circ f$ for standard function composition (“ g after f ”). So, in the statement of the theorem, $\nu_2 \circ \rho : O(\alpha) \rightarrow \mathbb{V} : x \mapsto \nu_2(\rho(x))$.

4.3 Variable renaming

In the proof of our main 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 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 14** (Renaming Lemma). *Let α be an FLIF expression and let θ be a permutation of \mathbb{V} . Then for every interpretation D , we have*

$$(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D \iff (\nu_1 \circ \theta, \nu_2 \circ \theta) \in \llbracket \theta(\alpha) \rrbracket_D.$$

4.4 Rewriting procedure

In order to be able to give a constructive proof of Theorem 13 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 15.** *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 $\text{var}(\alpha)$. There exists an io-disjoint FLIF 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 interpretation D and every valuation ν_1 , we have

$$\{\nu_2|_{O(\alpha)} \mid (\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D\} = \{\nu_2 \circ \rho \mid (\nu_1, \nu_2) \in \llbracket \beta \rrbracket_D\}.$$

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 5.

Terminology

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

Module access

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

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., $\text{var}(\alpha)$ 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.

Intersection

If α is of the form $\alpha_1 \cap \alpha_2$ then β equals $\beta_1 ; \gamma_1 ; \beta_2 ; \gamma_2 ; \eta$, where the constituent expressions are defined as follows.

- Let $W_1 = W \cup \text{var}(\alpha)$ and let ρ_1 be a renaming of $O(\alpha_1)$ that is an extension of ρ such that the image of $\rho_1 - \rho$ is disjoint from W_1 . By induction, there exists an io-disjoint rewriting of α_1 for W_1 and ρ_1 ; this yields β_1 .
- Let $W_2 = W_1 \cup O(\beta_1)$ and let ρ_2 be a bijection from $O(\alpha_2)$ to a set of variables that 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 $(\rho_1(y) = y)$ for $y \in O(\alpha_1) - O(\alpha_2)$. If $O(\alpha_1) - O(\alpha_2)$ is empty, γ_1 can be dropped from the expression; this qualification applies to similar situations below.
- γ_2 is defined symmetrically.
- η is the composition of all $(\rho_1(y) = \rho_2(y))$ for $y \in O(\alpha)$.

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 $W_2 = W \cup \text{var}(\alpha) \cup \rho(O(\alpha_1))$, 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 $W_1 = W \cup \text{var}(\alpha)$, 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 $\text{var}(\alpha) \cup O(\beta_2)$ 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 $W_1 = W \cup \text{var}(\alpha) \cup \rho(O(\alpha_2))$ 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 $W_2 = W \cup \text{var}(\alpha) \cup O(\beta_1)$ 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 $\text{var}(\alpha_2)$. We know $\text{var}(\alpha_2)$ 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 $W_1 = W \cup \text{var}(\alpha)$ 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 $\text{var}(\alpha_2)$ (which is nonempty by the same reasoning as given for the union case).
- η_2 is defined symmetrically.

4.5 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 13 that does not use variable assignment. Fix D to the interpretation where S is nonempty but T is empty. Then $\llbracket \alpha \rrbracket_D$ consists of all identical pairs of valuations. Take any valuation ν with $\nu(x) \neq \nu(x_1)$. Since $(\nu, \nu) \in \llbracket \alpha \rrbracket_D$, there should exist a valuation ν' with $\nu'(x_1) = \nu(x)$ such that $(\nu, \nu') \in \llbracket \beta \rrbracket_D$. 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.)

8:12 Input–Output Disjointness for FLIF

1. For every expression β without variable assignments, either $\llbracket \beta \rrbracket_D$ is empty, or $\llbracket \beta \rrbracket_D = \llbracket \gamma \rrbracket_D$ 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 $(\nu_1, \nu_2) \in \llbracket \gamma \rrbracket_D$, we have $\nu_1 = \nu_2$.

5 Correctness proof

We prove that β constructed by the method described in Section 4.4 satisfies the statement of Lemma 15. 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 interpretation D and $(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D$, we want to find ν such that $(\nu_1, \nu) \in \llbracket \beta \rrbracket_D$ and $\nu(\rho(y)) = \nu_2(y)$ for $y \in O(\alpha)$.

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

Below we present here the proofs for composition and union, as these are the clearest. The proofs for intersection and difference are more intricate and will appear in the journal version of this paper.

5.1 Composition

Henceforth, for any expression δ , we will use the notation $\nu_1 \xrightarrow{\delta} \nu_2$ to indicate that $(\nu_1, \nu_2) \in \llbracket \delta \rrbracket_D$.

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 as $\rho_1(O(\alpha_1))$ 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 $\text{var}(\alpha)$ 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 $O(\beta_2)$. In 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 $(\nu_1, \nu_2) \in \llbracket \alpha \rrbracket_D$, there exists ν such that $\nu_1 \xrightarrow{\alpha_1} \nu \xrightarrow{\alpha_2} \nu_2$. By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in \llbracket \beta_1 \rrbracket_D$ and $\nu_3(\rho_1(y)) = \nu(y)$ for $y \in O(\alpha_1)$. Also by induction, there exists ν_4 such that $(\nu, \nu_4) \in \llbracket \beta_2 \rrbracket_D$ and $\nu_4(\rho_2(y)) = \nu(y)$ for $y \in O(\alpha_2)$. By the Renaming Lemma (14), we have $(\nu \circ \theta, \nu_4 \circ \theta) \in \llbracket \theta(\beta_2) \rrbracket_D$.

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 $\text{var}(\alpha)$.
- Now let $u \in I(\alpha_2) - O(\alpha_1)$. Then $\theta(u) = u$, so $(\nu \circ \theta)(u) = \nu(u)$. Now since

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

and u belongs neither to $O(\alpha_1)$ nor to $O(\beta_1)$ (as $O(\beta_1)$ is disjoint from $\text{var}(\alpha)$), 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 determinacy and obtain ν_5 such that $(\nu_3, \nu_5) \in \llbracket \theta(\beta_2) \rrbracket_D$ and ν_5 agrees with $\nu_4 \circ \theta$ on $O(\beta_2)$. It follows that $(\nu_1, \nu_5) \in \llbracket \beta \rrbracket_D$, 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 $\text{var}(\alpha)$. 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 .

8:14 Input–Output Disjointness for FLIF

Second, assume $y \in O(\alpha_1) - O(\alpha_2)$. Since $(\nu_3, \nu_5) \in \llbracket \theta(\beta_2) \rrbracket_D$ 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 $(\nu, \nu_2) \in \llbracket \alpha_2 \rrbracket_D$ 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 so present for composition.

Since $(\nu_1, \nu_2) \in \llbracket \beta \rrbracket_D$, 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 $(\nu_1, \nu_3) \in \llbracket \alpha_1 \rrbracket_D$ and $\nu_3(y) = \nu(\rho_1(y))$ for $y \in O(\alpha_1)$. By the Renaming Lemma, we have $(\nu \circ \theta, \nu_2 \circ \theta) \in \llbracket \beta_2 \rrbracket_D$ (note that $\theta^{-1} = \theta$). By induction, there exists ν_4 such that $(\nu \circ \theta, \nu_4) \in \llbracket \alpha_2 \rrbracket_D$ 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 determinacy, there exists ν_5 such that $(\nu_3, \nu_5) \in \llbracket \alpha_2 \rrbracket_D$ and ν_5 agrees with ν_4 on $O(\alpha_2)$. It follows that $(\nu_1, \nu_5) \in \llbracket \alpha \rrbracket_D$, 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.

5.2 Union

Inputs

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

$$\begin{aligned} I(\gamma_i) &= O(\alpha_j) - O(\alpha_i) \\ O(\gamma_i) &= \rho(O(\alpha_j) - O(\alpha_i)) \end{aligned}$$

Note that $I(\gamma_i)$, being a subset of $\text{var}(\alpha)$, 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:

$$\begin{aligned} I(\beta_i; \gamma_i) &= I(\alpha_i) \cup (O(\alpha_j) - O(\alpha_i)) \\ O(\beta_i; \gamma_i) &= O(\beta_i) \cup \rho(O(\alpha_j)) \end{aligned}$$

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, $I(\eta_i) \subseteq \text{var}(\alpha_j)$. However, if $O(\beta_i)$ is empty then $O(\alpha_i)$ is too, so that $I(\alpha) = I(\alpha_i) \cup I(\alpha_j) \cup O(\alpha_j) = I(\alpha_i) \cup \text{var}(\alpha_j)$. 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

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

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

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

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 $\text{var}(\alpha)$. Then $\beta_i; \gamma_i$ is io-disjoint because both $O(\beta_i)$ and the image of ρ are disjoint from $\text{var}(\alpha)$. 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 $\text{var}(\alpha)$. 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 $(\nu_1, \nu_2) \in \llbracket \alpha_1 \rrbracket_D$; the reasoning for α_2 is analogous. By induction, there exists ν_3 such that $(\nu_1, \nu_3) \in \llbracket \beta_1 \rrbracket_D$ 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 $(\nu, \nu') \in \llbracket \delta \rrbracket_D$ (even independently of D).

Now let

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

8:16 Input–Output Disjointness for FLIF

so that $(\nu_1, \nu_5) \in \llbracket \beta \rrbracket_D$. 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 $\text{var}(\alpha)$.)

Soundness

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

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

By induction, there exists ν such that $(\nu_1, \nu) \in \llbracket \alpha_1 \rrbracket_D \subseteq \llbracket \alpha \rrbracket_D$ 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 $\rho(O(\alpha_1))$. Hence by (*) 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 $\text{var}(\alpha)$. 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 Conclusion

We have shown how to rewrite arbitrary FLIF expressions into io-disjoint ones. Our rewriting procedure is polynomial, as is readily verified from the description given in Section 4.4.

While the problem of “making programs io-disjoint” was, in this paper, interpreted in a specific manner according to the definitions of the FLIF language, there seems to be a more general quality about the problem that we have not been able to articulate. Of course, many examples of techniques that seem superficially similar can be cited, such as renaming bound variables in logic, or alpha-conversion in lambda calculus. But input and output as interpreted in this paper have a definite dynamic (one could say procedural) meaning which is lacking in such examples. So, while we are not aware about known results in logic or the foundations of programming languages that are technically related to our main result or our proof techniques, we would love to find out about them if they exist.

There are many interesting topics for further research on FLIF, and on LIF (Logic of Information Flows [17, 18, 1, 2]) in general. Some are already discussed in the cited papers; in closing this paper we list some more.

1. In our rewriting technique, there are really three, not two, categories of variables into play: inputs, outputs, and intermediate variables. It would be interesting to develop the framework further so that intermediate variables get a full treatment alongside inputs and outputs.

2. Investigate io-disjointness when FLIF is extended with additional operators, notably converse and transitive closure. We remark, however, that transitive closure seems to be at odds with io-disjointness. Indeed, if α is io-disjoint then the transitive closure of $\llbracket \alpha \rrbracket_D$ equals $\llbracket \alpha \rrbracket_D$ itself.
3. Delineate useful instances of LIF for which the problems of checking whether a variable is a semantic input, or output, are decidable.
4. Apply FLIF in practice. In this regard, the language could be made more practical by extending it with an explicit construct for hiding variables, as found, e.g., in process calculi [13], graph expressions [6], and graph logic [8].

References

- 1 H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska, and J. Van den Bussche. Executable first-order queries in the logic of information flows. In C. Lutz and J.C. Jung, editors, *Proceedings 23rd International Conference on Database Theory*, volume 155 of *Leibniz International Proceedings in Informatics*, pages 4:1–4:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.
- 2 H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska, and J. Van den Bussche. Inputs, outputs, and composition in the logic of information flows. In D. Calvanese, E. Erdem, and M. Thielscher, editors, *Proceedings 17th International Conference on Principles of Knowledge Representation and Reasoning*. IJCAI Organization, 2020.
- 3 H. Andr eka, S. Givant, and I. N emeti. *Decision problems for equational theories of relation algebras*, volume 126 of *Memoirs of the American Mathematical Society*. American Mathematical Society, 1997.
- 4 R. Angles, M. Arenas, P. Barcel o, A. Hogan, J. Reutter, and D. Vrgo c. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017.
- 5 R. Angles, P. Barcel o, and G. Rios. A practical query language for graph DBs. In L. Bravo and M. Lenzerini, editors, *Proceedings 7th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 1087 of *CEUR Workshop Proceedings*, 2013.
- 6 M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.
- 7 M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Morgan & Claypool, 2016.
- 8 L. Cardelli, Ph. Gardner, and G. Ghelli. A spatial logic for querying graphs. In P. Widmayer et al., editors, *Proceedings 29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.
- 9 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.
- 10 L. Libkin, W. Martens, and D. Vrgo c. Querying graph databases with XPath. In W.-C. Tan et al., editors, *Proceedings 16th International Conference on Database Theory*, pages 129–140. ACM, 2013.
- 11 V. Lifschitz. Formal theories of action (preliminary report). In J.P. McDermott, editor, *Proceedings 10th International Joint Conference on Artificial Intelligence*, pages 966–972. Morgan Kaufmann, 1987.
- 12 J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.
- 13 R. Milner. *Communicating and Mobile Systems: The π -calculus*. Cambridge University Press, 1999.
- 14 A. Nash and B. Lud ascher. Processing first-order queries under limited access patterns. In *Proceedings 23th ACM Symposium on Principles of Database Systems*, pages 307–318, 2004.

- 15 J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.
- 16 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.
- 17 E. Ternovska. Recent progress on the algebra of modular systems. In 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.
- 18 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.