First-Order Logic of Information Flows:

Input–output properties, Primitivity, Limited access patterns

Jan Van den Bussche

joint work with

Bart Bogaerts (Free U. Brussels)
Eugenia Ternovska (Simon Fraser U.)
Heba Aamer, Dimitri Surinx

UHASSELT

Logic of Information Flows [Ternovska]

Model complex systems by connecting modules

Module: a relation (input arguments, output arguments)

Connecting: first-order logic!

Also applications with higher-order relations, fixpoint logic

Dynamic semantics: "Law of Inertia"

Binary relation $Increment$

1st argument: input

2nd argument: output

Standard (static) semantics, assignment ν:

$$\nu \models Increment(x, y) \quad \Leftrightarrow \quad \nu(y) = \nu(x) + 1$$

Dynamic semantics, pair of assignments (ν_1, ν_2):

$$(\nu_1, \nu_2) \models Increment(x, y) \quad \Leftrightarrow \quad \nu_2(y) = \nu_1(x) + 1$$

and $\nu_2 = \nu_1$ elsewhere

Facebook example

Database D with a binary relation:

Friends

alice	bob
alice	carol
carol	dave
carol	eve

All pairs (ν_1, ν_2) such that $D, (\nu_1, \nu_2) \models Friends(x, y)$:

ν_1			ν_2		
x	y	z	x	y	z
alice	$*$	$-$	alice	bob	$-$
alice	$*$	$-$	alice	carol	$-$
carol	$*$	$-$	carol	dave	$-$

Example of a BAR!

Binary assignment relation (BAR)

BAR: set of pairs of variable assignments

The dynamic semantics of a relation is a BAR

Relations are atomic modules

Model complex modules by connecting them

Semantics of connectives: operations on BARs

E.g. $M_1(x, y) \vee M_2(u, v)$ union of BARs

Operations on BARs

Boolean connectives (union, intersection, difference)

Selection (equality)

Cylindrification (existential quantification)

Composition: $R \circ S = \{(\nu_1, \nu_3) \mid \exists \nu_2 : R(\nu_1, \nu_2) \wedge S(\nu_2, \nu_3)\}$

Converse: $R^{-1} = \{(\nu_2, \nu_1) \mid R(\nu_1, \nu_2)\}$

Operations on BARs

Boolean connectives (union, intersection, difference)

Selection (equality)

Cylindrification (existential quantification)

Composition: $R \circ S = \{(\nu_1, \nu_3) \mid \exists \nu_2 : R(\nu_1, \nu_2) \wedge S(\nu_2, \nu_3)\}$

Converse: $R^{-1} = \{(\nu_2, \nu_1) \mid R(\nu_1, \nu_2)\}$

Operations on BARs

Boolean connectives (union, intersection, difference)

Selection (equality)

Cylindrification (existential quantification)

Composition: $R \circ S = \{(\nu_1, \nu_3) \mid \exists \nu_2 : R(\nu_1, \nu_2) \wedge S(\nu_2, \nu_3)\}$

Converse: $R^{-1} = \{(\nu_2, \nu_1) \mid R(\nu_1, \nu_2)\}$

Operations on BARs

Boolean connectives (union, intersection, difference)

Selection (equality)

Cylindrification (existential quantification)

Composition: $R \circ S = \{(\nu_1, \nu_3) \mid \exists \nu_2 : R(\nu_1, \nu_2) \wedge S(\nu_2, \nu_3)\}$

Converse: $R^{-1} = \{(\nu_2, \nu_1) \mid R(\nu_1, \nu_2)\}$

Selection

$$\sigma^{l}_{x=y}(R) = \{(\nu_1, \nu_2) \in R \mid \nu_1(x) = \nu_1(y)\}$$

$$\sigma^{r}_{x=y}(R) = \{(\nu_1, \nu_2) \in R \mid \nu_2(x) = \nu_2(y)\}$$

$$\sigma^{lr}_{x=y}(R) = \{(\nu_1, \nu_2) \in R \mid \nu_1(x) = \nu_2(y)\}$$

Cylindrification

$$\exists_x^l(R) = \{(\nu_1', \nu_2) \mid \exists \nu_1 : (\nu_1, \nu_2) \in R \text{ and } \nu_1' = \nu_1 \text{ outside } x\}$$

$$\exists_x^r(R) = \{(\nu_1, \nu_2') \mid \exists \nu_2 : (\nu_1, \nu_2) \in R \text{ and } \nu_2' = \nu_2 \text{ outside } x\}$$

Example

$$\sigma^{r}_{x=y}(R) = R \cap \exists^{l}_{x} \sigma^{lr}_{x=y} \sigma^{lr}_{x=x} \exists^{l}_{x}(R)$$

op	ν_1		ν_2		condition
R	x_1	y_1	x_2	y_2	
$\exists^{l}_{x}$	$*$	y_1	x_2	y_2	
$\sigma^{lr}_{x=x}$	x_2	y_1	x_2	y_2	
$\sigma^{lr}_{x=y}$	x_2	y_1	x_2	y_2	$x_2 = y_2$
$\exists^{l}_{x}$	$*$	y_1	x_2	y_2	$x_2 = y_2$
$R \cap$	x_1	y_1	x_2	y_2	$x_2 = y_2$

The evaluation problem, first try

Expressions E built from relation names using the operators

Evaluation problem for expression E on instance D:

Input: An assignment ν_1

Output: All assignments ν_2 such that $D, (\nu_1, \nu_2) \models E$

Not practical...

- We should only need to give values for "input variables"
- We are only interested in values for "output variables"

What are the inputs, outputs of an expression?

Atomic modules (relations):

- input arguments are specified in the vocabulary
- remaining arguments are outputs

E.g. relation $Friend$ of input arity 1, total arity 2

Expression $Friend(x, y)$ has input var x, output var y

For complicated expressions, not so obvious

Semantic definition of an output variable

Variable x is an output of expression E if. . .

. . . there exists instance D, assignments ν_1, ν_2 such that

- $D, (\nu_1, \nu_2) \models E$
- $\nu_2(x) \neq \nu_1(x)$

Our definition of input variables

Variable x is an input of expression E if. . .

. . . there exists instance D, assignments ν_1, ν_2, ν_1' such that

- $D, (\nu_1, \nu_2) \models E$
- $\nu_1' = \nu_1$ except on x
- every ν_2' such that $D, (\nu_1', \nu_2') \models E$ differs from ν_2 on at least one output variable

LIF evaluation, ideal version

Input: Assignment ν_{in} on the input variables

Output: Projection on output variables of

$$\{\nu_{\mathsf{out}} \mid \exists \nu'_{\mathsf{in}} \supseteq \nu_{\mathsf{in}} : D, (\nu'_{\mathsf{in}}, \nu_{\mathsf{out}}) \models E\}$$

Unfortunately, deciding whether a variable is output (input) of some given expression is undecidable

(Reduction from satisfiability problem for first-order logic)

Syntactic approximation of inputs, outputs

E	$I(E)$	$O(E)$
$R(x,y)$	x	y
$R(x,y) - S(u,y)$	x,u	y
$R(x,y) - S(x,z)$	x,y,z	y
$R(x,y) \circ S(y,z)$	x	y,z
$\exists^l_x R(x,y)$	$\emptyset$	x,y
$\sigma^{lr}_{y=y}(R(x,y))$	x,y	$\emptyset$

We have definitions of $I(E)$ and $O(E)$ for any expression E

Compositional: $I(E_1 \text{ op } E_2)$ and $O(E_1 \text{ op } E_2)$ depend only on $I(E_j)$, $O(E_j)$, and op

Sound: $I(E)$ contains all semantic inputs, $O(E)$ all outputs

Optimal: Best possible compositional definition

Outline

What is LIF?

Inputs and outputs

Expressive power of LIF

- primitivity of composition

Forward LIF and limited access patterns

Is composition primitive?

1. Unlimited setting: infinite supply of variables

2. Bounded-variable: fixed, finite supply

3. Disjoint input–output setting

Composition is redundant under disjoint input–outputs

$$E_1 \circ E_2 \quad \equiv \quad \exists^r_{O_2}(E_1) \cap \exists^l_{O_1}(E_2)$$

- O_j is set of output variables for E_j
- Inputs, outputs of E_2 must be disjoint

Example for $R(x,y) \circ S(y,z)$:

op	ν_1			ν_2			condition
$\exists^r_z R(x,y)$	x_1	$*$	$*$	x_1	y_1	$*$	$R(x_1,y_1)$
$\exists^l_y S(y,z)$	x_2	$*$	$*$	x_2	y_2	z_2	$S(y_2,z_2)$
$\cap$	x_1	$*$	$*$	x_1	y_1	z_2	$R(x_1,y_1) \wedge S(y_1,z_2)$

Application to unlimited setting

To do $E_1 \circ E_2$:

1. Copy E_2's output variables to fresh variables
2. Composition becomes expressible
3. Copy back

$\Rightarrow$ Composition is not primitive

Composition is primitive in bounded-variable LIF

n variables

LIF is expressible in FO(3n)*

LIF w/o composition in FO(2n)

In LIF we can express existence of $3n$-clique

*For $n = 1$, LIF reduces to Tarski's algebra of binary relations

Outline

What is LIF?

Inputs and outputs

Expressive power of LIF

LIF and data with limited access patterns

- Executable first-order logic
- Forward LIF as an alternative

Data with limited access patterns

Relations can only be accessed
with values for input arguments

E.g. $Telephone(\underline{name}, number)$

Inspired by Data on the Web, Query Processing

Access plans based on relational algebra

Access join $E \underset{\substack{j_1=1 \\ \vdots \\ j_k=k}}{\bowtie} R$ for relation R of input arity k

Executable first-order logic

[Nash & Ludäscher 2004]

Syntactic restriction of first-order logic

Equivalent to relational algebra access plans

"Codd theorem" under limited access patterns

Is there room in the middle?

$$\text{Executable FO} \xleftarrow{\text{declarative}} \textbf{?} \xrightarrow{\text{procedural}} \text{Access Plans}$$

Forward LIF

$$
\begin{array}{rcl}
E & ::= & R(\bar{x};\bar{y}) \\
 & | & (x = y) \mid (x = c) \\
 & | & (x := y) \mid (x := c) \\
 & | & E \circ E \mid E \cup E \mid E \cap E \mid E - E
\end{array}
$$

Navigational graph query language

- nodes are variable assignments
- edges labeled by relation access
- equality tests, variable setting

Disjoint input–output Forward LIF $\equiv$ Executable FO

Conclusion

By giving first-order logic a dynamic semantics
with law of inertia. . .

. . . we obtain a declarative language for procedural knowledge