PρLog: a System for Rule-based Programming

Besik Dundua

Ilia Vekua Institute of Applied Mathematics
Ivane Javakhishvili Tbilisi State University
Tbilisi, Georgia

bdundua@gmail.com

Outline

Outline

What is PρLog

- A system that extends Prolog with strategic conditional transformation rules.
- Rules perform nondeterministic transformations of sequences.
- Strategies provide control on rule applications.
- PρLog system combines the power of logic programming and the flexibility of strategy-based conditional transformation in a single framework.

What is PρLog

- PρLog supports programming with four different types of variables: individual, sequence, function and context variables.
- PρLog is expressive enough to specifying and prototyping deductive systems, solvers for various equational theories, tools for XML querying and transformation, etc.
- PρLog code is usually quite short, declaratively clear, and reusable.
- Implemented in Prolog, available from

  http://www.risc.jku.at/˜tkutsia/software.html

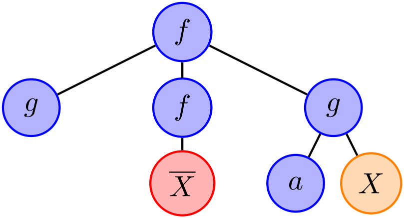
Different Kinds of Variables

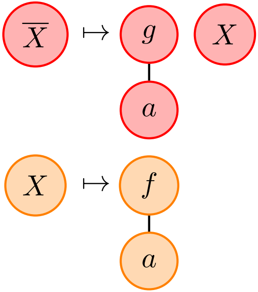
- Individual variables stand for single terms, while sequence variables stand for finite (possible empty) sequences of terms.
- Function variables denote function symbols, while context variables denote contexts that can be seen as unary functions with a single occurrence of the bound variable.
- Four different types of variables give the user flexibility on selecting subsequences in sequences or subterms/contexts in terms.
- This variables enhance expressive capabilities of a language, help to write short, neat, understandable code, and hide away many tedious data processing details from the programmer.

Intuition Behind Individual (X) and Sequence Variables ($\overline{X}$)

Example

$$f(g, f(\overline{X}), g(a, X)) \qquad \{\overline{X} \mapsto (g(a), X),\ X \mapsto f(a)\}$$

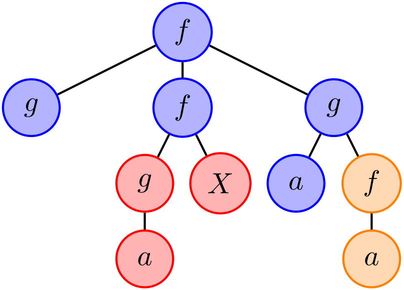


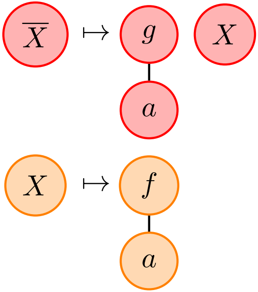


Intuition Behind Individual (X) and Sequence Variables ($\overline{X}$)

Example

$$f(g, f(g(a), y), g(a, f(a))) \quad \{\overline{X} \mapsto (g(a), X),\ X \mapsto f(a)\}$$

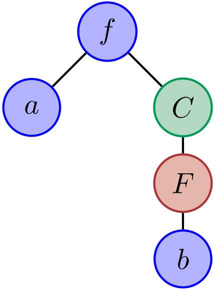


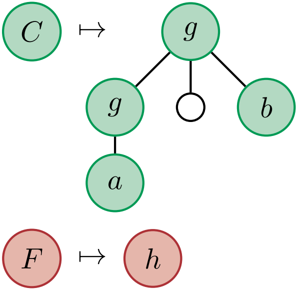


Intuition Behind Function (F) and Context Variables (C)

Example

$f(a, C(F(b)))$ $\{C \mapsto g(g(a), \circ, b),\ F \mapsto h\}$

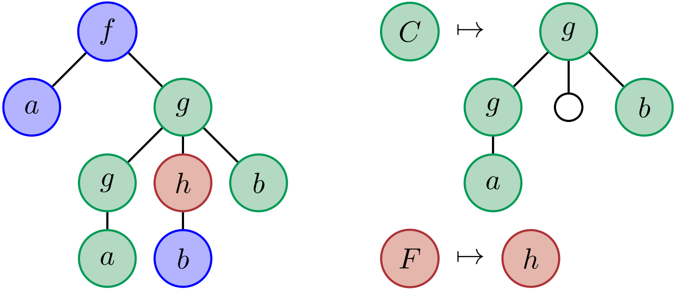




Intuition Behind Function (F) and Context Variables (C)

Example

$f(a, g(g(a), h(b), b))$ $\quad \{C \mapsto g(g(a), \circ, b),\ F \mapsto h\}$



Outline

Terms and Sequences

Terms and sequences are defined as follows:

- $t ::= X \mid f(s) \mid F(s) \mid C(t)$
- $s ::= t \mid \bar{X} \mid (s_1, \ldots, s_n)$

where

1. X is an individual variable
2. $\bar{X}$ is a sequence variable
3. F is a function variable
4. C is a context variable
5. f is a function symbol

Atoms, Literals

- ρ-atoms have a form $st :: s_1 \Rightarrow s_2$.
  - st: strategy (a term).
  - s_1, s_2: sequences.
  - Intuitive meaning: the strategy st transforms the sequence s_1 to the sequence s_2.
- Negation of a ρ-atom: $st :: s_1 \not\Rightarrow s_2$.
- Prolog conventions for naming symbols apply.

Clauses, Queries

- PρLog clauses have a form:
  $st :: s_1 \Rightarrow s_2 \text{ :- } L_1, \ldots, L_n,\ n \geq 0.$
- Each L_i is either a ρ-literal or a Prolog literal.
- Prolog clauses can be used in PρLog programs as well.
- PρLog queries: Conjunction of ρ- or Prolog literals:
  $L_1, \ldots, L_n$
- Well-modedness.

Built-in Strategies

- PρLog has a library of built-in strategies.

Built-in Strategies

- PρLog has a library of built-in strategies.
  - $id :: s_1 \Rightarrow s_2$ succeeds if s_1 matches with s_2.

Built-in Strategies

- PρLog has a library of built-in strategies.
  - $id :: s_1 \Rightarrow s_2$ succeeds if s_1 matches with s_2.
  - $choice(st_1, \ldots, st_n)$: nondeterministic choice.

Built-in Strategies

- PρLog has a library of built-in strategies.
  - $id :: s_1 \Rightarrow s_2$ succeeds if s_1 matches with s_2.
  - $choice(st_1, \ldots, st_n)$: nondeterministic choice.
  - $first\_one(st_1, \ldots, st_n)$, $n \geq 1$, selects the first st_i that does not fail and returns *only one result* of its application to the input sequence.

Built-in Strategies

- PρLog has a library of built-in strategies.
  - $id :: s_1 \Rightarrow s_2$ succeeds if s_1 matches with s_2.
  - $choice(st_1, \ldots, st_n)$: nondeterministic choice.
  - $first\_one(st_1, \ldots, st_n)$, $n \geq 1$, selects the first st_i that does not fail and returns *only one result* of its application to the input sequence.
  - $compose(st_1, \ldots, st_n)$, $n \geq 2$, first transforms the input sequence by st_1 and then transforms the result by $compose(st_2, \ldots, st_n)$.

Built-in Strategies

- PρLog has a library of built-in strategies.
  - $id :: s_1 \Rightarrow s_2$ succeeds if s_1 matches with s_2.
  - $choice(st_1, \ldots, st_n)$: nondeterministic choice.
  - $first\_one(st_1, \ldots, st_n)$, $n \geq 1$, selects the first st_i that does not fail and returns *only one result* of its application to the input sequence.
  - $compose(st_1, \ldots, st_n)$, $n \geq 2$, first transforms the input sequence by st_1 and then transforms the result by $compose(st_2, \ldots, st_n)$.
  - $nf(st)$ computes a normal form of the input hedge with respect to st. It never fails because if an application of st to a hedge fails, then st returns that sequence itself. Backtracking returns all normal forms.

Built-in Strategies

- PρLog has a library of built-in strategies.
  - $id :: s_1 \Rightarrow s_2$ succeeds if s_1 matches with s_2.
  - $choice(st_1, \dots, st_n)$: nondeterministic choice.
  - $first\_one(st_1, \dots, st_n)$, $n \geq 1$, selects the first st_i that does not fail and returns *only one result* of its application to the input sequence.
  - $compose(st_1, \dots, st_n)$, $n \geq 2$, first transforms the input sequence by st_1 and then transforms the result by $compose(st_2, \dots, st_n)$.
  - $nf(st)$ computes a normal form of the input hedge with respect to st. It never fails because if an application of st to a hedge fails, then st returns that sequence itself. Backtracking returns all normal forms.
  - $prox(\lambda) :: s_1 \Rightarrow s_2$ succeeds if s_1 matches approximately with s_2, at least with the degree λ.
  - etc.

Semantics of PρLog

We studied operational and declarative semantics of PρLog from constraint logic programming point of view.

Outline

Introduction

PρLog language

Programming in PρLog

Sorting

Example

The following program illustrates how bubble sort can be implemented in PρLog:

$$swap :: (\bar{X}, X, Y, \bar{Y}) \Rightarrow (\bar{X}, Y, X, \bar{Y}) \text{ :- } X > Y.$$
$$sort :: \bar{X} \Rightarrow \bar{Y} \text{ :- } nf(swap) :: \bar{X} \Rightarrow \bar{Y}.$$

Query:

$$sort :: (3, 1, 1, 2) \Rightarrow \bar{R}.$$

Outputs: $\bar{R} = (1, 1, 2, 3)$

Example: Merge Proximals

Assume our proximity relation is such that a and b are proximal with the degree 0.6 and b is close to c with the degree 0.8. Then we have:

- Merge proximals from a sequence:

$$merge\_proximals(\lambda) :: (\overline{X}, X, \overline{Y}, Y, \overline{Z}) \Longrightarrow (\overline{X}, \overline{Y}, Y, \overline{Z}) :- \\ \quad prox(\lambda) :: X \Longrightarrow Y.$$

$$merge\_all\_proximals(\lambda) := first\_one(nf(merge\_proximals(\lambda)))$$

Example: Merge Proximals

Assume our proximity relation is such that a and b are proximal with the degree 0.6 and b is close to c with the degree 0.8. Then we have:

- Merge proximals from a sequence:

$$
\begin{aligned}
&merge\_proximals(\lambda) :: (\overline{X}, X, \overline{Y}, Y, \overline{Z}) \Longrightarrow (\overline{X}, \overline{Y}, Y, \overline{Z}) :- \\
&\qquad prox(\lambda) :: X \Longrightarrow Y. \\
&merge\_all\_proximals(\lambda) := first\_one(nf(merge\_proximals(\lambda)))
\end{aligned}
$$

- Query:

$$
\begin{aligned}
&merge\_all\_proximals(0.5) :: (a, b, d, b, c) \Longrightarrow \overline{R}. \\
&\overline{R} = (d, c)
\end{aligned}
$$

Example: Merge Proximals

Assume our proximity relation is such that a and b are proximal with the degree 0.6 and b is close to c with the degree 0.8. Then we have:

- Merge proximals from a sequence:

  $$merge\_proximals(\lambda) :: (\overline{X}, X, \overline{Y}, Y, \overline{Z}) \Longrightarrow (\overline{X}, \overline{Y}, Y, \overline{Z}) :- \\ \quad prox(\lambda) :: X \Longrightarrow Y.$$

  $$merge\_all\_proximals(\lambda) := first\_one(nf(merge\_proximals(\lambda)))$$

- Query:

  $$merge\_all\_proximals(0.7) :: (a, b, d, b, c) \Longrightarrow \overline{R}.$$

  $$\overline{R} = (a, d, c)$$

Example: Merge Proximals

Assume our proximity relation is such that a and b are proximal with the degree 0.6 and b is close to c with the degree 0.8. Then we have:

- Merge proximals from a sequence:

$$merge\_proximals(\lambda) :: (\overline{X}, X, \overline{Y}, Y, \overline{Z}) \Longrightarrow (\overline{X}, \overline{Y}, Y, \overline{Z}) :- \\ \qquad prox(\lambda) :: X \Longrightarrow Y.$$

$$merge\_all\_proximals(\lambda) := \mathit{first\_one}(\mathit{nf}(merge\_proximals(\lambda)))$$

- Query:

$$merge\_all\_proximals(\mathit{0.5}) :: (b, d, b, c, a) \Longrightarrow \overline{R}.$$

$$\overline{R} = (d, c, a)$$

Example: Merge Proximals

Assume our proximity relation is such that a and b are proximal with the degree 0.6 and b is close to c with the degree 0.8. Then we have:

- Merge proximals from a sequence:

$$merge\_proximals(\lambda) :: (\overline{X}, X, \overline{Y}, Y, \overline{Z}) \Longrightarrow (\overline{X}, \overline{Y}, Y, \overline{Z}) :-$$
$$\qquad prox(\lambda) :: X \Longrightarrow Y.$$
$$merge\_all\_proximals(\lambda) := first\_one(nf(merge\_proximals(\lambda)))$$

- Query:

$$merge\_all\_proximals(0.7) :: (b, d, b, c, a) \Longrightarrow \overline{R}.$$
$$\overline{R} = (d, c, a)$$

Applications of PρLog

We have applications of PρLog in

- XML processing,
- Web reasoning,
- Implementing rewriting strategies,
- Extraction of frequent patterns from data mining workflows,
- Modeling of access control policies.

Acknowledgement

This work has been supported by the Shota Rustaveli National Science Foundation of Georgia under the grant YS-18-1480