

# P-log manual

August 27, 2019

## Contents

<b>1</b>	<b>System installation</b>	<b>2</b>
<b>2</b>	<b>System usage</b>	<b>2</b>
<b>3</b>	<b>Command Line Options</b>	<b>4</b>
<b>4</b>	<b>Syntax Description</b>	<b>4</b>
4.1	Sort definitions . . . . .	4
4.2	Attribute Declarations . . . . .	7
4.3	Program Statements . . . . .	7
4.3.1	Program Rules . . . . .	8
4.3.2	Pr-Atoms . . . . .	8
4.4	Program Examples . . . . .	8
<b>5</b>	<b>Error Checking</b>	<b>8</b>
5.1	Type errors . . . . .	8
5.1.1	Sort definition errors . . . . .	9
5.1.2	Attribute declarations errors . . . . .	11
5.1.3	Program statements errors . . . . .	11
5.2	Type warnings . . . . .	11

# 1 System installation

For the latest instructions on system installation, please refer to <https://github.com/iensen/plog2.0/wiki/Installation-Instructions>.

# 2 System usage

The system requires programs to be stored in ASCII files. The file can be of any extension (we recommend .pl or, not to be confused with Perl, .plog).

A P-log file acceptable by the system should consist of:

1. P-log program, consisting of:
  - sorts definitions,
  - attribute declarations, and
  - program rules.
2. Query.

For example, consider the following program:

sorts

```
#dice={d1,d2}.
#score={1,2,3,4,5,6}.
#person={mike,john}.
#bool = {true,false}.
```

attributes

```
roll:#dice->#score.
owns:#dice,#person->#bool.
```

statements

```
owns(d1,mike).
owns(d2,john).
```

```
random(roll(D)).
```

%probability information

```
pr(roll(D)=6|owns(D,mike))=1/4.
```

```
? roll(d1)=1.
```

The program, originally introduced in [3], describes a scenario with two dice being rolled, belonging to Mike and John respectively. The second die is more likely to produce 6 as an outcome, as defined by the pr-atom

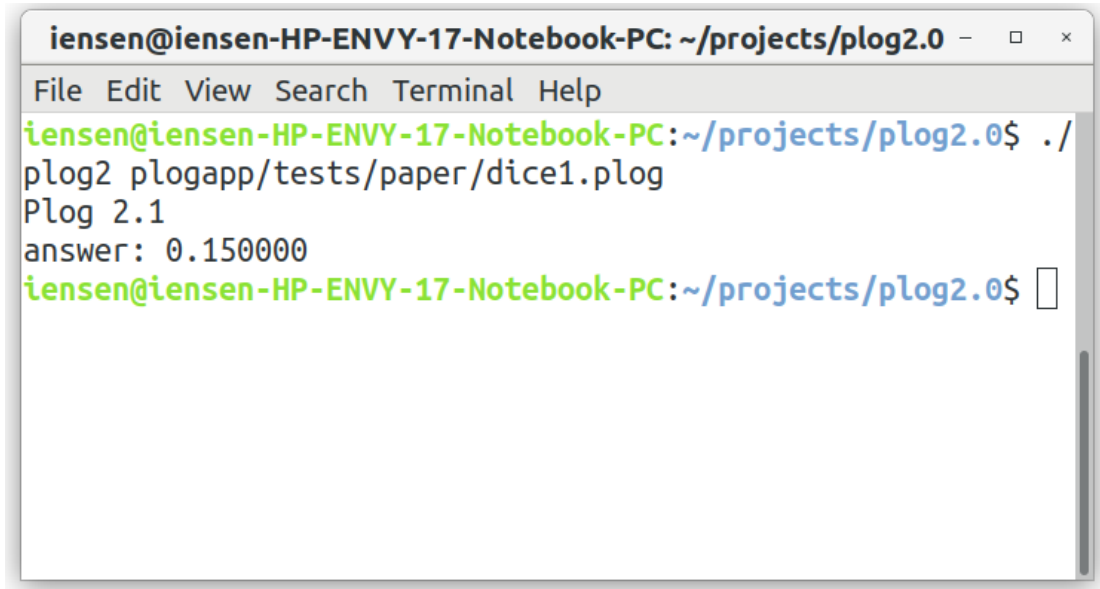
$$pr(roll(D) = 6 \mid owns(D, mike)) = 1/4.$$

By the so called *principle of indifference* used by P-log, the probabilities of the remaining outcomes equal to  $(1 - 1/4)/5 = 3/20$ . Thus, the answer to the query  $roll(d1) = 1$  is  $1/8$ . Note that all outcomes of the second die are equally likely, so the answer to the query  $roll(d2) = 1$  is equal to  $1/6$ .

To compute the query probability using the system, we need to store it in a file and run the command:

$plog2$  [path\_to\_file]

where *plog2* is the name of the p-log executable. For example, the file is stored in *plogapp/tests/paper/dice1.plog* in our system, and we get the following output:



```
iensen@iensen-HP-ENVY-17-Notebook-PC: ~/projects/plog2.0 - □ ×
File Edit View Search Terminal Help
iensen@iensen-HP-ENVY-17-Notebook-PC:~/projects/plog2.0$ ./
plog2 plogapp/tests/paper/dice1.plog
Plog 2.1
answer: 0.150000
iensen@iensen-HP-ENVY-17-Notebook-PC:~/projects/plog2.0$ □
```

Figure 1: P-log output

The details of the syntax of the language can be found in [2] and in the following sections.

### 3 Command Line Options

In this section we will describe the meanings of command line options supported by P-log. As of right now, the system only accepts a single argument which is a path to the p-log file:

- **input\_file** Specify the file where the sparc program is located.

See Section 2 for an example of using this argument.

### 4 Syntax Description

#### 4.1 Sort definitions

This section starts with a keyword *sorts* followed by a collection of sort definitions of the form:

$$\textit{sort\_name} = \textit{sort\_expression}.$$

*sort\_name* is an identifier preceeded by the pound sign (#). *sort\_expression* on the right hand side denotes a collection of strings called *a sort*. **As of right now, the system only supports a basic sort definition of the form  $\textit{sort\_name} = \{t_1, \dots, t_n\}$ , where  $t_1, \dots, t_n$  are ground terms. The remainder of the section is to be implemented in future..**

We divide all the sorts into *basic sorts* and *non-basic sorts*.

*Basic sorts* are defined as named collections of numbers and *identifiers*, i.e, strings consisting of

- letters:  $\{a, b, c, d, \dots, z, A, B, C, D, \dots, Z\}$
- digits:  $\{0, 1, 2, \dots, 9\}$
- underscore:  $\_$

and starting with a lowercase letter.

A *non-basic sort* also contains at least one *record* of the form  $\textit{id}(\alpha_1, \dots, \alpha_n)$  where *id* is an identifier and

$\alpha_1, \dots, \alpha_n$  are either identifiers, numbers or records.

We define sorts by means of expressions (in what follows sometimes referred to as statements) of six types:

1. **numeric range** is of the form:

$$number_1..number_2$$

where  $number_1$  and  $number_2$  are non-negative numbers such that  $number_1 \leq number_2$ .

The expression defines the set

of sequential numbers

$$\{number_1, number_1 + 1, \dots, number_2\}.$$

*Example:*

#sort1=1..3.

#sort1 consists of numbers  $\{1, 2, 3\}$ .

2. **identifier range** is of the form:

$$id_1..id_2$$

where  $id_1$  and  $id_2$  are identifiers both starting with a lowercase letter.

$id_1$  should be lexicographically<sup>1</sup> smaller than or equal to  $id_2$ , and the length of  $id_1$  must be less than or equal to the length of  $id_2$ . That is,  $id_1 \leq id_2$  and  $|id_1| \leq |id_2|$ .

The expression defines the set of strings  $\{s : id_1 \leq s \leq id_2 \wedge |id_1| \leq |s| \leq |id_2|\}$ .

*Example:*

#sort1=a..f.

#sort1 consists of letters  $\{a, b, c, d, e, f\}$ .

3. **set of ground terms** is of the form:

$$\{t_1, \dots, t_n\}$$

The expression denotes a set of *ground terms*  $\{t_1, \dots, t_n\}$ , defined as follows:

- numbers and identifiers are ground terms;
- If  $f$  is an identifier and  $\alpha_1, \dots, \alpha_n$  are ground terms, then  $f(\alpha_1, \dots, \alpha_n)$  is a ground term.

*Example:*

---

<sup>1</sup> The system default encoding is used for ordering of individual characters

#sort1={f(a), a, b, 2}.

4. **set of records** is of the form:

$$f(\text{sort\_name}_1(\text{var}_1), \dots, \text{sort\_name}_n(\text{var}_n)) : \text{condition}(\text{var}_1, \dots, \text{var}_n)$$

where  $f$  is an identifier, for  $1 \leq i \leq n$   $\text{sort\_name}_i$  occurs in one of the preceeding sort definitions and the condition on variables  $\text{var}_1, \dots, \text{var}_n$  (written as  $\text{condition}(\text{var}_1, \dots, \text{var}_n)$ ) is defined as follows:

- if  $\text{var}_i$  and  $\text{var}_j$  occur in the sequence  $\text{var}_1, \dots, \text{var}_n$  and  $\odot$  is an element of  $\{>, <, \leq, \geq\}$ , then  $\text{var}_i \odot \text{var}_j$  is a condition on  $\text{var}_1, \dots, \text{var}_n$ .
- if  $\mathcal{C}_1$  and  $\mathcal{C}_2$  are both conditions on  $\text{var}_1, \dots, \text{var}_n$ , and  $\oplus$  is an element of  $\{\cup, \cap\}$ , then  $(\mathcal{C}_1 \oplus \mathcal{C}_2)$  is a condition on  $\text{var}_1, \dots, \text{var}_n$ .
- if  $\mathcal{C}$  is a condition on  $\text{var}_1, \dots, \text{var}_n$ , then  $\text{not}(\mathcal{C})$  is also a condition on  $\text{var}_1, \dots, \text{var}_n$ .

Variables  $\text{var}_1, \dots, \text{var}_n$  occurring in parenthesis after sort names are optional as well as the condition  $\text{condition}(\text{var}_1, \dots, \text{var}_n)$ .

If a condition contains a subcondition  $\text{var}_i \odot \text{var}_j$ , then the sorts  $\text{sortname}_i$  and  $\text{sortname}_j$

must be defined by basic statements (the definition of a basic statement is given below after the definition of a concatenation statement).

The expression defines a collection of ground terms

$$\{f(t_1, \dots, t_n) : t_1 \in s_i \wedge \dots \wedge t_n \in s_n \wedge (\text{condition}(X_1, \dots, X_n)|_{X_1=t_1, \dots, X_n=t_n})\}$$

*Example*

#s=1..2.

#sf=f(s(X), s(Y), s(Z)) : (X=Y or Y=Z).

The sort #sf consists of records  $\{f(1, 1, 2), f(1, 1, 1), f(2, 1, 1)\}$

5. **set-theoretic expression** can be in one of the following forms:

- $\# \text{sort\_name}$
- an expression of the form (3), denoting a set of ground terms
- an expression of the form (4), denoting a set of records
- $(S_1 \nabla S_2)$ , where  $\nabla \in \{+, -, *\}$  and both  $S_1$  and  $S_2$  are set theoretic expressions

$\# \text{sort\_name}$  must be a name of a sort occurring in one of the preceeding sort definitions. The operations  $+$ ,  $*$  and  $-$  stand for union, intersection and difference correspondingly.

*Example :*

```
#sort1={a,b,2}.
#sort2={1,2,3} + {a,b,f(c)} + f(#sort1).
```

#sort2 consists of ground terms  $\{1, 2, 3, a, b, f(c), f(a), f(b), f(2)\}$ .

6. **concatenation** is of the form

$$[b\_stmt_1] \dots [b\_stmt_n]$$

$b\_stmt_1, \dots, b\_stmt_n$  must be *basic statements*, defined as follows:

- statements of the forms (1)-(3) are basic
- statement  $S$  of the form (5) is basic if:
  - it does not contain sort expressions of the form (4), denoting sets of records
  - none of curly brackets occurring in  $S$  contains a record
  - all sorts occurring in  $S$  are defined by basic statements

Note that basic statement can only define a basic sort.

*Example*<sup>2</sup>:

```
#sort1=[b] [1..100].
```

sort1 consists of identifiers  $\{b1, b2, \dots, b100\}$ .

## 4.2 Attribute Declarations

The second part of a P-log program starts with the keyword *attributes*

and is followed by statements of the form

$$attr\_symbol : \#sortName_1, \dots, \#sortName_n \rightarrow \#sortName$$

where  $attr\_symbol$  is an identifier (in what follows referred to as an attribute symbol) and  $\#sortName, \#sortName_1, \dots, \#sortName_n$  are sorts defined in sort definitions section of the program.

Multiple declarations containing the same attribute symbol are not allowed. Attributes with no arguments must be declared as  $attr\_symbol : sortName_1$  (example:  $a : \#bool$ ).

---

<sup>2</sup>We allow a shorthand 'b' for singleton set  $\{b\}$

### 4.3 Program Statements

The third part of a P-log program starts with the keyword *statements* followed by a collection *rules* and *pr-atoms* (in any order). Here we give a brief overview of the system assuming that a reader is familiar with notions of a term and a literal. For the details, please refer to [2].

#### 4.3.1 Program Rules

P-log rules are of the following form:

$$a(t) = y \leftarrow l_1, \dots, l_k, \text{not } l_{k+1} \dots \text{not } l_n. \quad (1)$$

where  $a(t) = y$  is a program atom and  $l_1, \dots, l_k$  are program literals. The atom  $a(t) = y$  is referred to as the *head* of the rule, and  $l_1, \dots, l_k$  is referred to as the *body* of the rule. A rule with  $k = 0$  is referred to as a *fact*. The system allows for a shorthand  $a(t)$  for  $a(t) = \text{true}$ , and standard arithmetic relations of the form  $t_1 \text{ op } t_2$ , where  $\text{op} \in \{>=, <=, >, <, =, !=\}$  in the body of the rule. Three special kinds of rules are *random selection rules*, *observations* and *actions*. As described in [2], random selection rule is a rule whose head is of one of the forms  $\text{random}(a, p)$  or  $\text{random}(a)$ . Action is a fact whose head is  $\text{do}(a, y)$ <sup>3</sup>. Observation is a fact whose head is of the form  $\text{obs}(a, y, t)$ .

#### 4.3.2 Pr-Atoms

Probability atoms (in short, pr-atoms) are of the form

$$\text{pr}(a(t) = y | B) = v.$$

where  $a(t) = y$  is a program atom,  $B$  is a collection of e-literals and  $v$  written as a fraction of the form  $n/m$ , where  $n$  and  $m$  are natural numbers. Example:

$$\text{pr}(\text{roll}(D)=6 \mid \text{not owns}(D, \text{mike}))=1/4.$$

### 4.4 Program Examples

For a collection of working P-log programs used for system testing and development, please refer to

<https://github.com/iensen/plog2.0/tree/master/plogapp/tests>.

---

<sup>3</sup>The system doesn't support rule labels and actions of the form  $\text{do}(r, a, y)$  described in [2]. Therefore there has to be at most one action of each attribute term. This is a limitation we plan to overcome in future.



## 5 Error Checking

Please be aware that some of the errors are not implemented yet.

Additional error checking is performed when no syntax error is found. Majority of the errors are related to types.

### 5.1 Type errors

Type errors are considered as serious issues which make it impossible to compile and execute the program. Type errors can occur in all four sections of a P-log program.

#### 5.1.1 Sort definition errors

The following are possible causes of a sort definition error that will result in a type error message from *SPARC*:

1. A set-theoretic expression (statement 5 in section 4.1) containing a sort name that has not been defined.

*Example:*

```
sorts
#s={a} .
#s2=#s1-#s .
```

2. Declaring a sort more than once.

*Example:*

```
sorts
#s={a} .
#s={b} .
```

3. An identifier range  $id_1..id_2$  (statement 2 in section 4.1) where  $id_1$  is greater than  $id_2$ .

*Example:*

```
sorts
#s=zbc..cbz .
```

4. A numeric range  $n_1..n_2$  (statement 1 in section 4.1) where  $n_1$  is greater than  $n_2$ .

*Example:*

```
sorts
#s=100500..1 .
```

5. A numeric range (statement 2 in section 4.1)  $n_1..n_2$  that contains an undefined constant.

*Example:*

```
#const n1=5.  
sorts  
#s=n1..n2.
```

6. An identifier range  $id_1..id_2$  (statement 3 in section 4.1) where the length of  $id_1$  is greater than the length of  $id_2$ .

*Example:*

```
sorts  
#s=abc..a.
```

7. A concatenation (statement 4 in section 4.1) that contains a non-basic sort.

*Example:*

```
sorts  
#s={ f (a) } .  
#sc=[a] [#s] .
```

8. A record definition (statement 5 in section 4.1) that contains an undefined sort.

*Example:*

```
sorts  
#s=1..2.  
#fs=f (s, s2) .
```

9. A record definition (statement 5 in section 4.1) that contains a condition with relation  $>$ ,  $<$ ,  $\geq$ ,  $\leq$  such that the corresponding sorts are not basic.

*Example:*

```
#s={ a, b } .  
#s1=f (#s) .  
#s2=g (s1 (X) , s2 (Y) ) : X>Y .
```

10. A variable that is used more than once in a record definition (statement 5 in section 4.1).

*Example:*

```

sorts
#s1={a} .
#s=f (#s1 (X) , #s1 (X) ) : (X!=X) .

```

11. A sort that contains an empty collection of ground terms.

*Example*

```

sorts
#s1={a,b,c}
#s=#s1-{a,b,c} .

```

### 5.1.2 Attribute declarations errors

1. An attribute with the same name is defined more than once. *Example:*

```

sorts
#s={a} .
attributes
a: #s -> #s .
a: #s, #s -> #s .

```

2. An attribute declaration contains an undefined sort. *Example:*

```

sorts
#s={a} .
attributes
p:#ss .

```

### 5.1.3 Program statements errors

In program rules and pr-atoms we first check each atom of the form  $a(t_1, \dots, t_n) = y$  and each term occurring in the program  $\Pi$  for satisfying the definitions of program atom and program term correspondingly (see Section 3 of [2]). In addition, we check that no sort occurs in a head of a rule of  $\Pi$ .

## 5.2 Type warnings

This section is to be completed and implemented. At the very least, we plan to support finite-domain constraint-based typechecking which will flag rules with no ground instances.

## References

- [1] Evgenii Balai, Michael Gelfond, and Yuanlin Zhang. Towards answer set programming with sorts. In *Logic Programming and Nonmonotonic Reasoning*, pages 135–147. Springer, 2013.
- [2] Evgenii Balai, Michael Gelfond, and Yuanlin Zhang. P-log: refinement and a new coherency condition. *Annals of Mathematics and Artificial Intelligence*, 86(1):149–192, Jul 2019.
- [3] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.