$\mathcal{ELPS}$ manual

March 21, 2014

Contents

1 System installation

For using the system, you need to have the following installed:

1. Java Runtime Environment (JRE):
   `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.
   The system was tested on Java versions 1.6.0_37 and 1.7.0_25.

2. The ELPS binary file:
   `https://github.com/iensen/elps/blob/master/elps.jar?raw=true`.

3. Clingo (version 4.2.1 or later):
   `http://sourceforge.net/projects/potassco/files/clingo/4.2.1`

Be sure the PATH system variable includes the directory where the clingo executable is located. For instructions on how to view/modify the PATH system variable, see either of the following links:
`http://www.java.com/en/download/help/path.xml`
`http://www.cyberciti.biz/faq/appleosx-bash-unix-change-set-path-environment-variable/`
To check if clingo is installed correctly, run the command `clingo -v`. See figure 1 for the expected output.

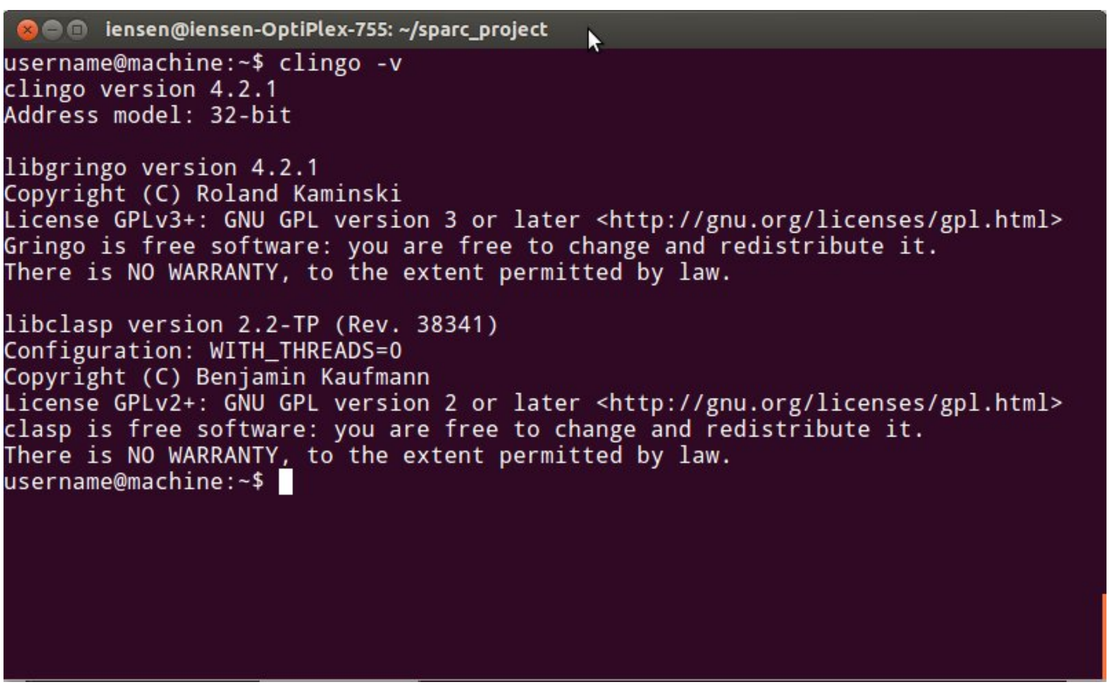

Figure 1: Checking the version of Clingo solver

2 System usage

To demonstrate the usage of the system we will use the program below described in [2].

```
sorts
#student = {mike,mary,ann}.

predicates

eligible(#student).
highGPA(#student).
fairGPA(#student).
minority(#student).
interview(#student).

rules
eligible(X):- highGPA(X).
eligible(X):- minority(X), fairGPA(X).
-eligible(X):- -fairGPA(X), -highGPA(X).
interview(X):- not K$ eligible(X), not K$ -eligible(X).

% data
fairGPA(mike) | highGPA(mike).
highGPA(mary).
```

To run $\mathcal{ELPS}$ solver on the program above, we change current directory to a directory having the file `program.sp` with the program written in it, and the downloaded file `elps.jar`. Then, we run the command:

```
> java -jar elps.jar program.sp
ELPS V1.03
program translated
World View 1 out of 1
{{eligible(mary), student(mary), student(mike), interview(mike),
 highGPA(mary), fairGPA(mike)}, {eligible(mary), student(mary),
 student(mike), interview(mike), highGPA(mike), highGPA(mary),
 eligible(mike)}}
```

The program has a single world view, which contains `interview(mike)` in both belief sets.

3 Syntax Description

3.1 Directives

Directives should be written before sort definitions, at the very beginning of a program. $\mathcal{ELPS}$ allows two types of directives:

#maxint

Directive #maxint specifies the maximum nonnegative number that could be used in arithmetic calculations. For example,

```
#maxint=15.
```

limits integers to [0,15].

#const

Directive #const allows one to define constant values. The syntax is:

```
#const constantName = constantValue.
```

where $constantName$ must begin with a lowercase letter and may be composed of letters, underscores and digits, and $constantValue$ is either a nonnegative number or the name of another constant defined before it.

3.2 Sort definitions

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

$$sort\_name = sort\_expression.$$

sort_name is an identifier preceeded by the pound sign (#). *sort_expression* on the right hand side denotes a collection of strings called $a\ sort$. 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: $\{a, b, c, d, ..., z, A, B, C, D, ..., Z\}$
- digits: $\{0, 1, 2, ..., 9\}$
- underscore: _

and starting with a lowercase letter.

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

$$n_1..n_2$$

   where n_1 and n_2 are non-negative integer numbers such that $n_1 \leq n_2$. The expression defines the set of sequential numbers $\{n_1, n_1 + 1, \ldots, n_2\}$.

   Example:

```
#sort1=1..3.
```

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

2. **identifier range** of the form

$$id_1..id_2$$

   where id_1 and id_2 are identifiers, id_1 is lexicographically [1] smaller than or equal to id_2, and the length of id_1 is 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** of the form

$$\{t_1, .., , t_n\}$$

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

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

   Example :

[1] The system default encoding is used for ordering of individual characters

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

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

$$f(sort\_name_1(var_1), ..., sort\_name_n(var_n)) : condition(var_1, ..., var_n)$$

where f is an identifier, for $1 \leq i \leq m$ $sort\_name_i$ occurs in one of the preceeding sort definitions and the condition on variables $var_1, ..., var_n$ (written as $condition(var_1, ..., var_n)$) is defined as follows:

   - if var_i and var_j occur in the sequence $var_1, ..., var_n$ and $\odot$ is an element of $\{>, <, \leq, \geq\}$, then $var_i \odot var_j$ is a condition on $var_1, ..., var_n$.
   - if $\mathcal{C}_1$ and $\mathcal{C}_2$ are both conditions on $var_1, ..., var_n$, and $\oplus$ is an element of $\{\cup, \cap\}$, then $(\mathcal{C}_1 \oplus \mathcal{C}_2)$ is a condition on $var_1, ..., var_n$.
   - if $\mathcal{C}$ is a condition on $var_1, ..., var_n$, then $not(\mathcal{C})$ is also a condition on $var_1, ..., var_n$.

Variables $var_1, ..., var_n$ occurring in parenthesis after sort names are optional as well as the condition :$condition(var_1, ..., var_n)$.

If a condition contains a subcondition $var_i \odot var_j$, then the sorts $sortname_i$ and $sortname_j$ 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
$\{f(t_1, \ldots, t_n) : t_1 \in s_i \wedge \cdots \wedge t_n \in s_n \wedge (condition(X_1, \ldots, X_n)|_{X_1=t_1,\ldots,X_n=t_n})\}$

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** in one of the following forms

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

$\#sort\_name$ 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** of the form

$$[b\_stmt_1]...[b\_stmt_n]$$

$b\_stmt_1, \ldots, b\_stmt_n$ 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[2]*.:*

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

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

3.3 Predicate Declarations

The second part of a $\mathcal{ELPS}$ program starts with the keyword $predicates$

and is followed by statements of the form

$$pred\_symbol(\#sortName_1, \ldots, \#sortName_n)$$

Where $pred\_symbol$ is an identifier (in what follows referred to as a predicate symbol) and $\#sortName_1$,...,$\#sortName_n$ are sorts defined in sort definitions section of the program.

Multiple declarations containing the same predicate symbol are not allowed. 0-arity predicates must be declared as $pred\_symbol()$. For any sort name $\#s$, the system includes declaration $\#s(\#s)$ automatically.

[2]We allow a shorthand 'b' for singleton set {b}

3.4 Program Rules

The third part of a $\mathcal{ELPS}$ program starts with the keyword *rules* followed by rules of the form

$$\ell_0|\ell_1 \dots |\ell_n \leftarrow g_1, \dots, g_m, not\ g_{m+1} \dots not\ g_k. \tag{1}$$

where $k \geq 0, m \geq 0, k \geq m$, each l_i is a literal and each g_i is either an extended literal(i.e, a literal possibly preceeded by not) or a subjective literal. Subjective literals can be in one of the forms $K\$\ell$,$M\$\ell$, $not\ K\$\ell$, or $not\ M\$\ell$, where ℓ is a literal.

Literals occurring in the heads of the rules must not be formed by predicate symbols occurring as sort names in sort definitions. In addition, rules must not contain *unrestricted variables*.

Definition 1 *(Unrestricted Variable) A variable occurrung in a rule of a $\mathcal{SPARC}$ program is called unrestriced if all its occurrences in the rule either belong to some relational atoms of the form $term1$ **rel** $term2$ (where **rel** $\in \{>, >=, <, <=, =, !=\}$) and/or some term appearing in a head of a choice or aggregate element.*

Example 1 Consider the following $\mathcal{ELPS}$ program:

```
sorts
#s={f(a),b}.
predicates
p(#s).
rules
p(f(X)):-Y<2,2=Z,F>3,#count{Q:Q<W,p(W),T<2},p(Y).
```

Variables F,T,Z,Q are unrestricted.

4 Typechecking

If no syntax errors are found, a static check of the program is performed. Any type-related problems found during this check will be output as type errors

4.1 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 $\mathcal{ELPS}$ program.

4.1.1 Sort definition errors

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

1. A set-theoretic expression (statement 5 in section 3.2) 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 3.2) 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 3.2) where n_1 is greater than n_2.

   Example:

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

5. A numeric range (statement 1 in section 3.2) $n_1..n_2$ that contains an undefined constant.

   Example:

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

6. An identifier range $id_1..id_2$ (statement 2 in section 3.2) where the length of id_1 is greater than the length of id_2.

   Example:

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

7. A concatenation (statement 6 in section 3.2) that contains a non-basic sort.

   Example:

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

8. A record definition (statement 5 in section 3.2) that contains an undefined sort.

   Example:

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

9. A record definition (statement 5 in section 3.2) that contains a condition with relation $>, <, \geq, \leq$ such that the corresponding sorts are not defined by basic statements.

   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 3.2).

    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}.
```

4.1.2 Predicate declarations errors

1. A predicate with the same name is defined more than once.

   Example:

   ```
   sorts
   #s={a}.
   predicates
   p(#s).
   p(#s,#s).
   ```

2. A predicate declaration contains an undefined sort.

   Example:

   ```
   sorts
   #s={a}.
   predicates
   p(#ss).
   ```

4.1.3 Program rules errors

In program rules we first check each atom of the form $p(t_1, \ldots, t_n)$ and each term occurring in the program Π for satisfying the definitions of program atom and program term correspondingly[1]. Moreover, we check that no sort occurs in a head of a rule of Π.

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] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *The Journal of Logic Programming*, 19:73–148, 1994.