

All you need to know to run your solver in **probo**, with specific reference to ICCMA'15

Federico Cerutti Nir Oren Hannes Straß Matthias Thimm Mauro Vallati
Serena Villata

Rev. 5 — 18 November 2014

1 Introduction

This document describes the interfaces that a solver needs to export in order to be executed by **probo**. For a description of **probo** and of the ICCMA'15 competition, the reader is invited to study (Cerutti *et al.*, 2014) and to regularly check the official website <http://argumentationcompetition.org>.

Knowledge of (Cerutti *et al.*, 2014) is a prerequisite for this document.



Remark for the ICCMA'15 Competition


These boxes highlight what is **necessary** to know for the **forthcoming ICCMA'15 competition**.

2 Terminology

In this document, we consider the following semantics (Baroni *et al.*, 2011) for Dung's argumentation framework (Dung, 1995) supported by **probo**:

- conflict-freeness (hereafter **CF**);
- admissibility (hereafter **ADM**);
- complete (hereafter **CO**);
- grounded (hereafter **GR**);
- preferred (hereafter **PR**);
- stable (hereafter **ST**);
- stage (hereafter **STG**);
- semi-stable (hereafter **SST**);

- ideal (hereafter ID);
- CF2 (hereafter CF2).



ICCMA'15

Semantics considered in ICCMA'15

- complete (hereafter CO);
- grounded (hereafter GR);
- preferred (hereafter PR);
- stable (hereafter ST).

The problems supported by **probo** discussed in this document are:

- Decision problems:
 1. Credulous acceptance (hereafter DC);
 2. Skeptical acceptance (hereafter DS);
- Enumeration problems:
 1. **all** the extensions (hereafter EE);
 2. **some** extension (hereafter SE)¹;
 3. enumerate all the arguments that are credulously inferred (hereafter EC)¹;
 4. enumerate all the arguments that are skeptically inferred (hereafter ES)¹.


ICCMA'15

Problems considered in ICCMA'15


- Decision problems:
 1. Credulous acceptance (hereafter DC);
 2. Skeptical acceptance (hereafter DS);
- Enumeration problems:
 1. **all** the extensions (hereafter EE);
 2. **some** extension (hereafter SE)¹.

probo supports the following three serialisations (see (Cerutti *et al.*, 2014) for examples):

- Aspartix format (hereafter **apx**);

¹For single-status semantics, this problem is equivalent to EE.

- CNF format (hereafter `cnf`);
- Trivial graph format (hereafter `tgf`).



ICCMA'15

File formats considered in ICCMA '15

- Aspartix format (hereafter `apx`);
- CNF format (hereafter `cnf`);
- Trivial graph format (hereafter `tgf`).

3 Interfaces to `probo`: Reference Guide

`probo` interacts with each solver in two ways. First, it queries each solver's capabilities in terms of supported file types and problems. Secondly, it invokes the solver on a specific argumentation framework for a specific purpose (problem).

Each solver must write — on the standard output — the answers to the invocation. There are three types of answers:

1. YES or NO — for decision problems;
2. [`e11`, `e12`, `e13`] — list of elements (e.g. a list of arguments for credulous/skeptical enumeration problems);
3. [[`e11`, `e12`, `e13`], [...], ...] — list of lists of elements (e.g. extensions enumeration);

3.1 Capabilities query

Each solver to be invoked by `probo` must export the following options for query purposes:

- when invoked without option, the solver must write author and version information to the standard output;

- `--formats`

when invoked with this parameter the solver must write the list of supported file types to the standard output. Acceptable output are any sublists of:

[`apx`, `cnf`, `tgf`]

- `--problems`

when invoked with this parameter the solver must write the list of supported problems to the standard output. Acceptable output are any sublists of:

[DC-CF, DC-ADM, DC-CO, DC-GR, DC-PR, DC-ST, DC-STG, DC-SST, DC-ID, DC-CF2, DS-CF, DS-ADM, DS-CO, DS-GR, DS-PR, DS-ST, DS-STG, DS-SST, DS-ID, DS-CF2, EC-CF, EC-ADM, EC-CO, EC-GR, EC-PR, EC-ST, EC-STG, EC-SST, EC-ID, EC-CF2, ES-CF, ES-ADM, ES-CO, ES-GR, ES-PR, ES-ST, ES-STG, ES-SST, ES-ID, ES-CF2, EE-CF, EE-ADM, EE-CO, EE-GR,

EE-PR, EE-ST, EE-STG, EE-SST, EE-ID, EE-CF2, SE-CF, SE-ADM, SE-STG, SE-SST, SE-ID, SE-CF2, SE-CO, SE-GR, SE-PR, SE-ST]



Problems considered in ICCMA '15

[DC-CO, DC-GR, DC-PR, DC-ST, DS-CO, DS-GR, DS-PR, DS-ST, EE-CO, EE-GR, EE-PR, EE-ST, SE-CO, SE-GR, SE-PR, SE-ST]

For example, if a solver supports credulous acceptance for complete semantics and grounded semantics, and enumeration of stable extensions, the expected output is like

[DC-CO, DC-GR, EE-ST]

(the order in the list does not matter).

3.2 Query Problem's Answer

Each solver to be invoked by `probo` must be able to parse and respond to the following options:

- `-f filename` — giving the input file name for a problem;
- `-fo format` — e.g. `-f apx` for specifying that `filename` is in Aspartix format;
- `-p problem` — e.g. `-p DC-PR` for specifying that the problem to be solved is the credulous acceptance w.r.t. preferred semantics;
- `-a additional` — providing additional problem related information. E.g. `-a a1` for specifying that the argument to be checked for credulous acceptance is `a1`.

The syntactically acceptable outputs depend on the type of problem:

- for decision problems, i.e. DC or DS:
valid output are either YES or NO ;
- for enumeration of some extension, enumeration of the arguments that are credulously/skeptically inferred, i.e. SE, EC or ES:
valid output is a list of arguments (e.g. [a1, a2]);
- for extensions enumeration:
valid output is a list of lists of arguments (e.g. [[a1,a2],[a3]]). Please note that this is the case also for single status semantics. In the case no extension exists, the answer must be an empty list [].

4 A tutorial

In this section, we describe, step-by-step, the procedure for allowing `probo` to invoke a generic solver. For simplicity we consider `ArgSemSATv0.2` (<http://sourceforge.net/projects/argsemsat/>) as the solver.

Since ArgSemSATv0.2 does not export the interfaces requested by `probo`, we can create a bash script acting as a proxy (Listing 1).

Listing 1: Bash Script for ArgSemSATv0.2

```
1 #!/bin/bash
2 solver="ArgSemSAT"
3
4 fileinput=""
5 problem=""
6 format=""
7
8 if [ "$#" = "0" ]
9 then
10     echo "ArgSemSATv0.2"
11     echo "Federico Cerutti <federico.cerutti@acm.org>"
12     echo "Mauro Vallati <mauro.vallati@hud.ac.uk>"
13 fi
14
15 while [ "$1" != "" ]; do
16     case $1 in
17         "--formats")
18             echo '[apx]'
19             exit 0
20         ;;
21         "--problems")
22             echo '[EE-PR]'
23             exit 0
24         ;;
25         "-p")
26             shift
27             problem=$1
28         ;;
29         "-f")
30             shift
31             fileinput=$1
32         ;;
33         "-fo")
34             shift
35             format=$1
36         ;;
37         esac
38     shift
39 done
40
41 if [ "$format" = "apx" -a "$problem" = "EE-PR" ];
42 then
43     res=$(($(dirname $0)/$solver $fileinput -ExtSAT GLUCOSE -sem preferred-df)
44
45     echo -n "["
46     echo $res | sed 's/{/[/g' | sed 's/}/]/g' | tr -d '\n' | sed 's/}{/},{/g' |
47     sed 's/\ /,/g'
48     echo "]"
49 fi
```

First of all, if the script is invoked without arguments (lines 8–13), then information regarding version and authors is provided.

ArgSemSATv0.2 can work on files in Aspartix format only, therefore the return value of the proxy script when it is invoked with the parameter `--formats` is `[apx]` (lines 17–20).

Then, since ArgSemSATv0.2 can be tested for performance purposes only on a specific problem, viz. the enumeration of preferred extensions, the return values of the proxy script when it is invoked with the parameter `--problems` must be [EE-PR] (lines 21–24).

The bash script accepts the parameters `-p` (lines 24–28), `-f` (lines 29–32), and `-fo` (lines 33–36) storing accordingly the provided data.

Finally, if invoked on a EE-PR problem with a `apx` type file as input, at lines 41–48 the bash script invokes the actual program ArgSemSAT (line 43) and it stores the result of the computation in the variable `res`. Then, the output is formatted accordingly to `probo`'s requirements (line 46) by simple string transformation and printed to the standard output.

If you want to test the result, download `probo` from <http://sourceforge.net/projects/probo/>, add your solver and your bash script interface to the `solvers/` directory, and add your bash script to the list of solvers in the configuration file. For instance, this is a portion of the configuration file listing two solvers, viz. Tweety and ArgSemSATv0.2:

```
solvers:
- solvers/tweetysolver-v1.0.8.sh
- solvers/argsemsatv0.2.sh
```

5 A Configurable Bash Script

As part of the `probo` distribution, the following bash script is also provided (Listing 2). It can be customised by providing:

1. a `information` procedure displaying author and version (lines 27–33);
2. a `solver` procedure which invokes the actual program (lines 36–64);
3. a `parse_output` procedure which parses the output of the actual program in order to adhere to `probo`'s requests;
4. the list of accepted input type (lines 95–97, please comment the unsupported formats);
5. the list of accepted problems (lines 100–163, please comment the unsupported problems).

This bash script is provided for solvers' developer convenience only, who are clearly free to implement `probo`'s interface in other ways.

Listing 2: Generic Bash Script

```
1 #!/bin/bash
2 # (c)2014 Federico Cerutti <federico.cerutti@acm.org> --- MIT LICENCE
3 # Generic script interface to probo http://sourceforge.net/projects/probo/
4 # Please feel free to customize it for your own solver
5
6
7 # function for echoing on standard error
8 echoerr()
9 {
10 # to remove standard error echoing, please comment the following line
11 echo "$@" 1>&2;
12 }
13
14 #####
15 # C O N F I G U R A T I O N
```

```

16 #
17 # this script must be customized by defining:
18 # 1) procedure for printing author and version information of the solver
19 # (function "information")
20 # 2) suitable procedures for invoking your solver (function "solver");
21 # 3) suitable procedures for parsing your solver's output
22 # (function "parse_output");
23 # 4) list of supported format (array "formats");
24 # 5) list of supported problems (array "problems").
25
26 # output information
27 function information
28 {
29     # example for ArgSemSATv0.2
30     echo "ArgSemSATv0.2"
31     echo "Federico Cerutti <federico.cerutti@acm.org>"
32     echo "Mauro Vallati <mauro.vallati@hud.ac.uk>"
33 }
34
35 # how to invoke your solver: this function must be customized
36 function solver
37 {
38     fileinput=$1 # input file with correct path
39
40     format=$2 # format of the input file (see below)
41
42     problem=$3 # problem to solve (see below)
43
44     additional=$4 # additional information, i.e. name of an argument
45
46
47     # dummy output
48     echoerr "input file: $fileinput"
49     echoerr "format: $format"
50     echoerr "problem: $problem"
51     echoerr "additional: $additional"
52
53     # example for ArgSemSATv0.2
54     if [ "$format" = "apx" -a "$problem" = "EE-PR" ];
55     then
56         ./${(dirname $0)}/ArgSemSAT $fileinput -ExtSAT GLUCOSE -sem preferred-df
57     else
58         echoerr "unsupported format or problem"
59         exit 1
60     fi
61 }
62
63
64 # how to parse the output of your solver in order to be compliant with proba:
65 # this function must be customized
66 # proba accepts solutions of the form:
67 # [arg1,arg2,...,argN] for
68 #
69 # 1. some extension enumeration (SE)
70 # 2. enum. arguments credulously inferred (EC)
71 # 3. enum. arguments skeptically inferred (ES)
72 # [[arg1,arg2,...,argN],[...],...] for extension(s) enumeration
73 # YES/NO for decision problems
74 function parse_output
75 {

```

```

75 problem=$1
76 output="$2"
77
78 echoerr "original output: $output"
79
80 #example of parsing for ArgSemSATv0.2, which returns "{arg1,arg2,...}\n{...}\n..."
81 if [ "$problem" = "EE-PR" ];
82 then
83     echo -n "["
84     echo $output | sed 's/{/[/g' | sed 's/}/]/g' | tr -d '\n' \
85     | sed 's/}{/},{/g' | sed 's/\ /,/g'
86     echo "]"
87 else
88     echoerr "unsupported format or problem"
89     exit 1
90 fi
91 }
92
93 # accepted formats: please comment those unsupported
94 formats[1]="apx" # "aspartix" format
95 formats[2]="cnf" # conjunctive normal form
96 formats[3]="tgf" # trivial graph format
97
98 # problems that can be accepted: please comment those unsupported
99
100 #+-----+
101 #|          I C C M A   '1 5   L I S T   O F   P R O B L E M S           |
102 #|                                                                 |
103 problems[1]="DC-CO" # Decide credulously according to Complete semantics
104 problems[2]="DC-GR" # Decide credulously according to Grounded semantics
105 problems[3]="DC-PR" # Decide credulously according to Preferred semantics
106 problems[4]="DC-ST" # Decide credulously according to Stable semantics
107 problems[5]="DS-CO" # Decide skeptically according to Complete semantics
108 problems[6]="DS-GR" # Decide skeptically according to Grounded semantics
109 problems[7]="DS-PR" # Decide skeptically according to Preferred semantics
110 problems[8]="DS-ST" # Decide skeptically according to Stable semantics
111 problems[9]="EE-CO" # Enumerate all the extensions according to Complete
112 semantics
113 problems[10]="EE-GR" # Enumerate all the extensions according to Grounded
114 semantics
115 problems[11]="EE-PR" # Enumerate all the extensions according to Preferred
116 semantics
117 problems[12]="EE-ST" # Enumerate all the extensions according to Stable
118 semantics
119 problems[13]="SE-CO" # Enumerate some extension according to Complete
120 semantics
121 problems[14]="SE-GR" # Enumerate some extension according to Grounded
122 semantics
123 problems[15]="SE-PR" # Enumerate some extension according to Preferred
124 semantics
125 problems[16]="SE-ST" # Enumerate some extension according to Stable semantics
126 #|
127 #| E N D   O F   I C C M A   '1 5   L I S T   O F   P R O B L E M S   |
128 #+-----+
129
130 problems[17]="DC-ADM" # Decide credulously according to admissibility
131 problems[18]="DC-CF2" # Decide credulously according to CF2 semantics
132 problems[19]="DC-CF" # Decide credulously according to conflict-freeness

```



```

126 problems [20]="DC-ID" # Decide credulously according to Ideal semantics
127 problems [21]="DC-SST" # Decide credulously according to Semi-stable semantics
128 problems [22]="DC-STG" # Decide credulously according to Stage semantics
129 problems [23]="DS-ADM" # Decide skeptically according to admissibility
130 problems [24]="DS-CF2" # Decide skeptically according to CF2 semantics
131 problems [25]="DS-CF" # Decide skeptically according to conflict-freeness
132 problems [26]="DS-ID" # Decide skeptically according to Ideal semantics
133 problems [27]="DS-SST" # Decide skeptically according to Semi-stable semantics
134 problems [28]="DS-STG" # Decide skeptically according to Stage semantics
135 problems [29]="EC-ADM" # Enumerate all the arguments credulously inferred
    according to admissibility
136 problems [30]="EC-CF2" # Enumerate all the arguments credulously inferred
    according to CF2 semantics
137 problems [31]="EC-CF" # Enumerate all the arguments credulously inferred
    according to conflict-freeness
138 problems [32]="EC-CO" # Enumerate all the arguments credulously inferred
    according to Complete semantics
139 problems [33]="EC-GR" # Enumerate all the arguments credulously inferred
    according to Grounded semantics
140 problems [34]="EC-ID" # Enumerate all the arguments credulously inferred
    according to Ideal semantics
141 problems [35]="EC-PR" # Enumerate all the arguments credulously inferred
    according to Preferred semantics
142 problems [36]="EC-SST" # Enumerate all the arguments credulously inferred
    according to Semi-stable semantics
143 problems [37]="EC-STG" # Enumerate all the arguments credulously inferred
    according to Stage semantics
144 problems [38]="EC-ST" # Enumerate all the arguments credulously inferred
    according to Stable semantics
145 problems [39]="EE-ADM" # Enumerate all the extensions according to
    admissibility
146 problems [40]="EE-CF2" # Enumerate all the extensions according to CF2
    semantics
147 problems [41]="EE-CF" # Enumerate all the extensions according to conflict-
    freeness
148 problems [42]="EE-ID" # Enumerate all the extensions according to Ideal
    semantics
149 problems [43]="EE-SST" # Enumerate all the extensions according to Semi-stable
    semantics
150 problems [44]="EE-STG" # Enumerate all the extensions according to Stage
    semantics
151 problems [45]="ES-ADM" # Enumerate all the arguments skeptically inferred
    according to admissibility
152 problems [46]="ES-CF2" # Enumerate all the arguments skeptically inferred
    according to CF2 semantics
153 problems [47]="ES-CF" # Enumerate all the arguments skeptically inferred
    according to conflict-freeness
154 problems [48]="ES-CO" # Enumerate all the arguments skeptically inferred
    according to Complete semantics
155 problems [49]="ES-GR" # Enumerate all the arguments skeptically inferred
    according to Grounded semantics
156 problems [50]="ES-ID" # Enumerate all the arguments skeptically inferred
    according to Ideal semantics
157 problems [51]="ES-PR" # Enumerate all the arguments skeptically inferred
    according to Preferred semantics
158 problems [52]="ES-SST" # Enumerate all the arguments skeptically inferred
    according to Semi-stable semantics
159 problems [53]="ES-STG" # Enumerate all the arguments skeptically inferred
    according to Stage semantics

```

```

160 problems[54]="ES-ST" # Enumerate all the arguments skeptically inferred
    according to Stable semantics
161 problems[55]="SE-ADM" # Enumerate some extension according to admissibility
162 problems[56]="SE-CF2" # Enumerate some extension according to CF2 semantics
163 problems[57]="SE-CF" # Enumerate some extension according to conflict-freeness
164 problems[58]="SE-ID" # Enumerate some extension according to Ideal semantics
165 problems[59]="SE-SST" # Enumerate some extension according to Semi-stable
    semantics
166 problems[60]="SE-STG" # Enumerate some extension according to Stage semantics
167
168 # E N D   O F   C O N F I G U R A T I O N   S E C T I O N
169 #####
170
171 function list_output
172 {
173     declare -a arr=("${!1}")
174     check_something_printed=false
175     echo -n '['
176     for i in ${arr[@]};
177     do
178         if [ "$check_something_printed" = true ];
179         then
180             echo -n ", "
181         fi
182         echo -n $i
183         check_something_printed=true
184     done
185     echo ']'
186 }
187
188 function main
189 {
190     if [ "$#" = "0" ]
191     then
192         information
193         exit 0
194     fi
195
196     local local_problem=""
197     local local_fileinput=""
198     local local_format=""
199     local local_additional=""
200
201     while [ "$1" != "" ]; do
202         case $1 in
203             "--formats")
204                 list_output formats[@]
205                 exit 0
206             ;;
207             "--problems")
208                 list_output problems[@]
209                 exit 0
210             ;;
211             "-p")
212                 shift
213                 local_problem=$1
214             ;;
215             "-f")
216                 shift

```

```

217     local_fileinput=$1
218     ;;
219     "-fo")
220     shift
221     local_format=$1
222     ;;
223     "-a")
224     shift
225     local_additional=$1
226     ;;
227     esac
228     shift
229 done
230
231 res=$(solver $local_fileinput $local_format $local_problem $local_additional)
232
233 parse_output $local_problem "$res"
234 }
235
236 main $@
237 exit 0

```

References

- Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.
- Federico Cerutti, Nir Oren, Hannes Strass, Matthias Thimm, and Mauro Vallati. The First International Competition on Computational Models of Argumentation (ICCMA’15) — Supplementary Notes on probo. <http://sourceforge.net/p/probo/code/HEAD/tree/trunk/doc/>, 2014.
- P. M. Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2):321–358, 1995.