`solved-af` — Solved Argumentation Framework

an abstract argumentation framework solver to learn from.

```
$  solved-af -p DC-ST -a arg1 -f instance.tgf -fo tgf
```

This repository contains the implementation of an argumentation framework solver to be submitted as part of the final year project (Bachelor Thesis) of David Simon Tetruashvili @ King's College London.

`solved-af...`

- ...is SAT-reduction based.
- ...is written in pure Python 3.
- ...supports both inputs formats defined in the ICCMA'19 specification (TGF, and APX).
- ...is capable of solving all classic track problems under the Dung's four main abstract argumentation semantics.

Prerequisites

`solved-af` uses `glucode-syrup` SAT solver. `glucode-syrup` has to be installed in the `PATH` for `solved-af` to work out of the box.

It is possible to change the SAT solver command from within the source code via the `SAT_COMMAND` constant variable in `saf/tasks.py` provided that the new solver has similar DIMACS input/output formats.

Installation

Via `install.sh` script

Use the included `install.sh` script to install `solved-af`.

```
$ git clone https://github.kcl.ac.uk/K1764158/solved-af.git
$ cd solved-af
$ chmod +x ./install.sh && ./install.sh
```

Via `pip`

Use the package manager `pip` to install `solved-af`.

```
$ git clone https://github.kcl.ac.uk/K1764158/solved-af.git
$ cd solved-af
$ pip install -e .
```

Usage

`solved-af` follows the established ICCMA solver interface closely with the added option of input validation via the `-v` flag.

```
usage: solved-af [ -h ] -p TASK -f INPUTFILE -fo {tgf, apx}
                        [ -a QUERYARGUMENT ]
                        [ --formats][ --problems][ -v ]

required arguments:
  -p TASK, --problemTask TASK
  Argrumentation framework problem task to solve
  -f INPUTFILE, --inputFile INPUTFILE
  Path to file containing an argumentation framework encoding
  -fo {tgf, apx}, --fileFormat {tgf, apx}
  Input file format

optional arguments:
  -a QUERYARGUMENT, --argument QUERYARGUMENT
  Argument to check acceptance for
  --formats             List all supported input file formats and exit
  --problems            List all supported problems tasks and exit
  -v, --validate        Validate the input file before parsing
```

Extendability

In contrast to other available AF solvers, `solved-af` is meant to be flexible, easy to understand and extend (albeit at the cost of performance). For example, here is how to use a reduction parser (`TheoryParser`) to reduce stable semantics to SAT and subsequently solve the full enumeration (`EE-ST`) task under it.

```python
# (theories.py)
# Create SAF CNF theory templates which encode an argumentation
# semantics (e.g., stable).

def stable_in_theory_template(a, f):
    clause = [attacker for attacker in f.getAttackersOf(a)]
    clause.append(a)
    return [clause]


def stable_out_theory_template(a, f):
    return [[-attacker, -a]
            for attacker in f.getAttackersOf(a)]


# Create a SAT reduction parser instance which uses the
```

```
# above templates.

stable_theories = CNFTheory.fromTemplates(
        stable_in_theory_template, stable_out_theory_template)

stableLabellingParser = DIMACSParser(*stable_theories,
                                     vars_per_argument=1)

# ------------------------------------------------------------ #

# (tasks.py)
# Use the reduction parser to solve AF problems.

def stableFullEnumeration(framework):
    return fullEnumeration(framework, stableLabellingParser)
```

Testing scripts

Included in this repository are some scripts used for running `solved-af` on a set of instances and comparing them to a set of reference results. Both of these data sets can be found here. These scripts rely on `comp-exts-mpz` (O. Rodrigues) and `runsolver` which are not included here.

Running `./run-tests.sh` or `./run_decision_tests.sh` without arguments shows usage messages.

The scripts will generate CSV files with the recorded data. They are compatible with any ICCMA interface compatible solver.

Licence

GNU GPL v3