Enumerating Preferred Extensions Using
ASP Domain Heuristics: The ASPrMin
Solver

Wolfgang FABER a, Mauro VALLATI b, Federico CERUTTI c and
Massimiliano GIACOMIN d

a Alpen-Adria-Universität Klagenfurt, Austria
b University of Huddersfield, UK
c Cardiff University, UK
d Università degli Studi di Brescia, Italy

Abstract. This paper briefly describes the solver ASPrMin, which enumerates preferred extensions and scored first in the Extension Enumeration problem—the only one implemented—of the Preferred Semantics Track of the Second International Competition on Computational Models of Argumentation, ICCMA17.

Keywords. argumentation, solver, ASP

1. Abstract Argumentation and Preferred Extensions

We recall some basic notions in abstract argumentation (cf. [2]).

An argumentation framework (AF) is a pair Γ = (A, R) where A is a set of arguments and R ⊆ A × A. We say that b attacks a iff (b, a) ∈ R, also denoted as b → a. The set of attackers of an argument a will be denoted as a− ≜ \{b : b → a\}, the set of arguments attacked by a will be denoted as a+ ≜ \{b : a → b\}.

Given an AF Γ = (A, R), a set S ⊆ A is a conflict–free set of Γ if \[a, b ∈ S \text{ s.t. } a → b; \] an argument a ∈ A is acceptable with respect to a set S ⊆ A of Γ if \[∀b ∈ A \text{ s.t. } b → a, ∃c ∈ S \text{ s.t. } c → b; \] a set S ⊆ A is an admissible set of Γ if S is a conflict–free set of Γ and every element of S is acceptable with respect to S of Γ. A set S ⊆ A is a preferred extension of Γ, i.e. \(S ∈ E_{PR}(Γ)\), if S is a maximal (w.r.t. ⊆) admissible set of Γ.

2. Implementation Using ASP Solver clingo

We use a straightforward and well-known encoding for admissible extensions, see [3,1]. Given an AF Γ = (A, R), for each a ∈ A a fact arg(a), is created and for

\[1\text{https://helios.hud.ac.uk/scommv/storage/ASPrMin-v1.0.tar.gz}\]
each \((a, b) \in \mathcal{R}\) a fact \(\text{att}(a, b)\) is created (this corresponds to the apx file format in the ICCMA competition). Together with the program

\[
\begin{align*}
\text{in}(X) :&\quad \neg \text{not out}(X), \text{arg}(X). \\
\text{out}(X) :&\quad \neg \text{in}(X), \text{arg}(X). \\
\text{defeated}(X) :&\quad \neg \text{in}(Y), \text{att}(Y, X). \\
\text{not_defended}(X) :&\quad \neg \text{att}(Y, X), \neg \text{defeated}(Y). \\
&\quad \neg \text{in}(X), \text{in}(Y), \text{att}(X, Y). \\
&\quad \neg \text{in}(X), \text{not_defended}(X).
\end{align*}
\]

we form \(\text{admasp}_1\) and there is a one-to-one correspondence between answer sets of \(\text{admasp}_1\) and admissible extensions.

We can then exploit domain heuristics in the ASP solver \(\text{clasp}\), a component of \(\text{clingo}\) [5]. Following [6,4], command line option \(--\text{heuristic=Domain}\) enables domain heuristics, and \(--\text{dom-mod}=3,16\) applies modifier true to all atoms that are shown. Since we want to apply the modifier to all atoms with predicate \text{in}, we augment \(\text{admasp}_1\), by the line \#show in/1. This means that the solver heuristics will prefer atoms with predicate \text{in} over all other atoms and will choose these atoms as being true first. This will find a subset maximal answer sets with respect to predicate \text{in}. The system \(\text{clingo}\) also allows for solution recording, see [4], by specifying command line option \(--\text{enum-mod=domRec}\). Together with the domain heuristic, this will enumerate all subset maximal answer set with respect to predicate \text{in}.

\text{ASPrMin} essentially makes the following call and does some minor post-processing using a shell script:

\texttt{clingo admasp1 \textcolor{blue}{\text{\textbackslash --heuristic}}=Domain \textcolor{blue}{\text{\textbackslash --dom-mod}}=3,16 \textcolor{blue}{\text{\textbackslash --enum-mod}}=domRec

Acknowledgements

We would like to thank Stefan Woltran and Torsten Schaub for pointing us to this way of implementing subset maximality.

References