argmat-sat: Applying SAT Solvers for Argumentation Problems based on Boolean Matrix Algebra*

Fuan Pu, Hang Ya, and Guiming Luo

School of Software, Tsinghua University, Beijing, China
Pu.Fuan@gmail.com, yah16@mails.tsinghua.edu.cn,
gluo@tsinghua.edu.cn

Abstract. This paper presents a system description of `argmat-sat`, which chooses SAT solvers to encode and compute argumentation problems. We describe the encoding approaches for `argmat-sat` based on Boolean matrix algebra, and an assumption-based algorithm for computing the reasoning tasks of the maximal semantics (e.g., `PR` and `ID`) and the maximal range semantics (e.g., `SST` and `STG`).

1 CNF encodings for argumentation semantics

Dung's argumentation semantics [1] can be encoded into Boolean constraint models based on Boolean matrix algebra [2]. These constraint models can be solved by SAT solvers, whose inputs are usually represented in CNF.

Let $\Delta = \langle \mathcal{X}, \mathcal{R} \rangle$ be a finite abstract *argumentation framework* (AF) with $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$. Table 1 provides the conversions from some of the Boolean constraint models for the basic semantics (conflict-free, admissible and complete semantics) into CNF encodings, in which the bold letters $\mathbf{x}$, $\mathbf{o}$ and $\mathbf{p}$ are three $n \times 1$ Boolean variable vectors, indexing the arguments in $\mathcal{X}$. The Boolean variable vector $\mathbf{x}$ is the Boolean vector representation of an extension (w.r.t. $\mathcal{X}$). If $\mathbf{x}_i = 1$, it indicates x_i is in the extension; otherwise, x_i is not in the extension. The Boolean variable vectors $\mathbf{o}$ and $\mathbf{p}$ consist of auxiliary variables, and they also have specific meanings. The corresponding argument set of $\mathbf{o}$ (w.r.t. $\mathcal{X}$) is exactly the argument set that is attacked by the argument set corresponding to $\mathbf{x}$. In addition, $\mathbf{p} = \neg\mathbf{o}$ corresponds to the result of neutrality function with respect to $\mathbf{x}$ (see [2]). The encoding for `ST` can be used for solving stable semantics, and the encodings for `AD` and `CO` can be used for searching admissible, complete, preferred, grounded and ideal extensions. The CNF encodings of [`ST1`] and [`CO1`] for stable and complete semantics have also appeared in [3]. The CNF encoding of [`AD1`] is also introduced in [4]. For the maximal range semantics, i.e., the semi-stable semantics (`SST`) and the stage semantics (`STG`), we introduce another auxiliary variable vector $\mathbf{r}$, which represents the range with respect to the extension $\mathbf{x}$. The CNF encodings of the two semantics are shown in Table 2.

Note that Table 1 and Table 2 merely give the CNF encoding for one argument x_i. The CNF encoding for the semantics σ (w.r.t. Δ) is the conjunction of all CNF

* This work was supported by the Fund NSFC61572279.

Table 1. The conversation for basic semantics

σ	Boolean-matrix-algebra-based encoding		CNF encoding	*Vars*
ST	[ST2]	$\mathbf{x} = \mathcal{N}(\mathbf{x})$	$\mathcal{H}_{[\texttt{ST2}]}(x_i) = \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{x}_i \vee \neg \mathbf{x}_j), \Big(\mathbf{x}_i \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \mathbf{x}_j\Big)$	n
AD	[AD1] with [CF2]	$\begin{cases} \mathbf{x} \leqslant \mathcal{N}(\mathbf{x}) \\ \mathcal{R}^-(\mathbf{x}) \leqslant \mathcal{R}^+(\mathbf{x}) \end{cases}$	$\mathcal{H}_{[\texttt{AD1}]}(x_i) = \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{x}_i \wedge \mathbf{x}_j), \bigwedge_{x_k \in \mathcal{R}^-(x_i)} \Big(\neg \mathbf{x}_k \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \mathbf{x}_j\Big)$	n
	[AD2]	$\begin{cases} \mathbf{x} \leqslant \mathcal{N}(\mathbf{x}) \\ \mathbf{x} \leqslant \mathcal{F}(\mathbf{x}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{p} = \mathcal{N}(\mathbf{x}) \\ \mathbf{x} \leqslant \mathbf{p} \\ \mathbf{x} \leqslant \mathcal{N}(\mathbf{p}) \end{cases}$	$\mathcal{H}_{[\texttt{AD2}]}(x_i) = \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{p}_i \vee \neg \mathbf{x}_j), \Big(\mathbf{p}_i \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \mathbf{x}_j\Big), (\neg \mathbf{x}_i \vee \mathbf{p}_i), \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{x}_i \vee \neg \mathbf{p}_j).$	$2n$
CO	[CO1] with [CF2]	$\begin{cases} \mathbf{x} \leqslant \mathcal{N}(\mathbf{x}) \\ \mathbf{x} = \mathcal{F}(\mathbf{x}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{o} = \mathcal{R}^+(\mathbf{x}) \\ \mathbf{x} \leqslant \neg \mathbf{o} \\ \mathbf{x} = \mathcal{N}(\neg \mathbf{o}) \end{cases}$	$\mathcal{H}_{[\texttt{CO1}]}(x_i) = \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\mathbf{o}_i \vee \neg \mathbf{x}_j), \Big(\neg \mathbf{o}_i \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \mathbf{x}_j\Big), (\neg \mathbf{x}_i \vee \neg \mathbf{o}_i), \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{x}_i \vee \mathbf{o}_j), \Big(\mathbf{x}_i \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \neg \mathbf{o}_j\Big).$	$2n$
	[CO2]	$\mathbf{x} = \mathcal{N}(\mathbf{x}) * \mathcal{F}(\mathbf{x}) \Leftrightarrow \begin{cases} \mathbf{p} = \mathcal{N}(\mathbf{x}) \\ \mathbf{x} = \mathcal{N}(\mathbf{x}) * \mathcal{N}(\mathbf{p}) \end{cases}$	$\mathcal{H}_{[\texttt{CO2}]}(x_i) = \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{p}_i \vee \neg \mathbf{x}_j), \Big(\mathbf{p}_i \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \mathbf{x}_j\Big), (\neg \mathbf{x}_i \vee \mathbf{p}_i) \wedge \bigwedge_{x_j \in \mathcal{R}^+(x_i)} (\neg \mathbf{x}_i \vee \neg \mathbf{p}_j), \Big(\neg \mathbf{p}_i \vee \mathbf{x}_i \vee \bigvee_{x_j \in \mathcal{R}^+(x_i)} \mathbf{p}_j\Big).$	$2n$

encodings of all arguments in Δ, i.e.,

$$\mathcal{H}_\sigma^\Delta(\mathbf{x}) = \bigwedge_{x_i \in \mathcal{X}} \mathcal{H}_\sigma(x_i) \tag{1}$$

Table 1 and Table 2 also give the total number of variables that are used in each CNF encoding (see the rightmost column of the table). By using these CNF encodings, we can easily solve the reasoning problems for stable and complete semantics based on an incremental SAT solver. For a semantics, different CNF encodings may have different performances, since they use various number of variables and clauses. The research in this regard is still in empirical test.

2 Computing the preferred semantics under assumption space

Since the maximal requirement (w.r.t. $\subseteq$) in preferred semantics is hard to be encoded by pure Boolean formulas, in the current implementation of `argmat-sat`, we propose

Table 2. The conversation for the maximal range semantics

σ	**Boolean-matrix-algebra-based encoding**		**CNF encoding**	*Vars*
SST	[SST1] (using [AD1])	$\begin{cases}[\texttt{AD1}]\\ \mathbf{r}=\mathbf{x}+\mathcal{R}^+(\mathbf{x})\end{cases}$	$\mathcal{H}_{[\texttt{SST1}]}(x_i)=\mathcal{H}_{[\texttt{AD1}]}(x_i)$, $\Big(\neg\mathbf{r}_i\vee\mathbf{x}_i\vee\bigvee_{x_j\in\mathcal{R}^+(x_i)}\mathbf{x}_j\Big)$, $\big(\neg\mathbf{x}_i\vee\mathbf{r}_i\big)\wedge\bigwedge_{x_j\in\mathcal{R}^+(x_i)}\big(\neg\mathbf{x}_j\vee\mathbf{r}_i\big)$.	$2n$
	[SST2] (using [AD2] or [CO2])	$\begin{cases}[\texttt{AD2}]\text{ or }[\texttt{CO2}]\\ \mathbf{r}=\mathbf{x}+\neg\mathbf{p}\end{cases}$	$\mathcal{H}_{[\texttt{SST2}]}(x_i)=\mathcal{H}_{[\texttt{AD2}]}(x_i)$ or $\mathcal{H}_{[\texttt{CO2}]}(x_i)$, $\big(\neg\mathbf{r}_i\vee\mathbf{x}_i\vee\neg\mathbf{p}_i\big)$, $\big(\neg\mathbf{x}_i\vee\mathbf{r}_i\big)\wedge\big(\mathbf{p}_i\vee\mathbf{r}_i\big)$.	$3n$
	[SST3] (using [CO1])	$\begin{cases}[\texttt{CO1}]\\ \mathbf{r}=\mathbf{x}+\mathbf{o}\end{cases}$	$\mathcal{H}_{[\texttt{SST3}]}(x_i)=,\mathcal{H}_{[\texttt{CO1}]}(x_i)$, $\big(\neg\mathbf{r}_i\vee\mathbf{x}_i\vee\mathbf{o}_i\big)$, $\big(\neg\mathbf{x}_i\vee\mathbf{r}_i\big)\wedge\big(\neg\mathbf{o}_i\vee\mathbf{r}_i\big)$.	$3n$
STG	[STG1] (using [CF2])	$\begin{cases}\mathbf{x}\leqslant\mathcal{N}(\mathbf{x})\\ \mathbf{r}=\mathbf{x}+\mathcal{R}^+(\mathbf{x})\end{cases}$	$\mathcal{H}_{[\texttt{STG1}]}(x_i)=\bigwedge_{x_j\in\mathcal{R}^+(x_i)}(\neg\mathbf{x}_i\wedge\mathbf{x}_j)$, $\Big(\neg\mathbf{r}_i\vee\mathbf{x}_i\vee\bigvee_{x_j\in\mathcal{R}^+(x_i)}\mathbf{x}_j\Big)$, $\big(\neg\mathbf{x}_i\vee\mathbf{r}_i\big)\wedge\bigwedge_{x_j\in\mathcal{R}^+(x_i)}\big(\neg\mathbf{x}_j\vee\mathbf{r}_i\big)$.	$2n$

a novel approach to compute preferred semantics by introducing assumption space to SAT solvers. Algorithm 1 shows the main process of the current implementation to enumerate all preferred extensions, where the assumption space is used as a temporary clause base for searching a maximal extension. When a maximal extension is found, the clauses in assumption space are erased in order to correctly search the next maximal extension. The idea of this algorithm is also utilized to compute the ideal semantics.

3 Computing the maximal range semantics

The implementations for computing the semi-stable and stage semantics are also based on the assumption space. The enumerating tasks for the two maximal range semantics can be described as below:

(i) Use the assumption-space-based approach to search a maximal range $\mathbf{max\_r}$ and reset the assumption space to empty;
(ii) Fix the range by adding the clause $(\mathbf{r}=\mathbf{max\_r})$ to the assumption space, then enumerate all extensions under this maximal range;
(iii) Reset the assumption space to empty again, and then repeat step (i) and (ii) until all ranges are found and all extensions are enumerated.

It can be seen that the process for computing the maximal range semantics uses the assumption space twice, one for finding a maximal range, and another one for enumerating all extensions under the range.

Algorithm 1 Enumerate all PR extensions

Require: $\Delta = \langle \mathcal{X}, \mathcal{R} \rangle$ — input an AF;
Ensure: $\mathcal{E}_{\mathtt{PR}}(\Delta)$ — return all PR extensions of Δ;

$\mathcal{E}_{\mathtt{PR}}(\Delta) \leftarrow \emptyset$;
`solver` $\leftarrow$ **new SATSolver**($\mathcal{H}^{\Delta}_{\mathtt{PR}}(\mathbf{x})$); ▷ Create the SAT solver
while ($\mathbf{s} \leftarrow$ `solver`.$hasSolution()$ & $\mathbf{s}! = \text{null}$) **do** ▷ Decide and get a solution
    $\mathbf{max\_s} \leftarrow$ null; ▷ If `solver` has a solution, it must have a maximal solution
    repeat ▷ The internal loop for searching a maximal solution
        $\mathbf{max\_s} \leftarrow \mathbf{s}$; ▷ Store the currently maximal solution
        `solver`.$addAssumptionClause(\ \mathbf{s} \leqslant \mathbf{x}\ \&\ \neg(\mathbf{s} = \mathbf{x})\ )$;
        ▷ Add a clause to the assumption space of `solver` for ensuring a maximal solution
    until ($\mathbf{s} \leftarrow$ `solver`.$hasSolution()$ & $\mathbf{s}! = \text{null}$)
        ▷ When not find a larger solution, the internal loop terminates
    $\mathcal{E}_{\mathtt{PR}}(\Delta) \leftarrow \mathcal{E}_{\mathtt{PR}}(\Delta) \cup \mathbf{max\_s}$; ▷ $\mathbf{max\_s}$ is a maximal solution and is added into $\mathcal{E}_{\mathtt{PR}}(\Delta)$
    `solver`.$RemoveAllAssumptionClauses()$; ▷ Erase all clauses in assumption space
    `solver`.$addHardClause(\ \neg(\mathbf{x} \leqslant \mathbf{max\_s})\ )$
    ▷ Ban the solutions that subsets $\mathbf{max\_s}$ (in hard space), and continue to the next search
end while
return $\mathcal{E}_{\mathtt{PR}}(\Omega)$;

4 Some implementation details

`argmat-sat` is implemented by C++, supports all computational problems of ICCMA-2017, and satisfies the standard command interface of the requirements of ICCMA-2017. Its SAT engine is selected as CryptoMiniSat5[1], which provides an easy-to-use programming interface with high computational efficiency, won the incremental track at SAT Competition 2016, and gets 3rd place at the parallel track. The current version of `argmat-sat`, submitted to ICCMA-2017, supports multi-threading, and can be successfully compiled and run under both Windows and Unix OS. The source codes can be found on the website of our project `argumatrix`[2].

References

1. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Journal of Artificial Intelligence **77**(2) (September 1995) 321–357
2. Fuan, P., Guiming, L., JIANG, Z.: Encoding argumentation semantics by boolean algebra. IEICE Transactions on Information and Systems **100**(4) (2017) 838–848
3. Lagniez, J.M., Lonca, E., Mailly, J.G.: CoQuiAAS: A constraint-based quick abstract argumentation solver. In: Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on, IEEE (2015) 928–935
4. Wallner, J.P., Weissenbacher, G., Woltran, S.: Advanced SAT techniques for abstract argumentation. In: International Workshop on Computational Logic in Multi-Agent Systems, Springer (2013) 138–154

[1] `https://github.com/msoos/cryptominisat`
[2] `https://sites.google.com/site/argumatrix/`