ABSTRACT ARGUMENTATION

Answer Set Programming Encodings for Argumentation Frameworks

Sarah Gaggl

Dresden, ICCL Summer School 2017
Motivation

- Argumentation Frameworks provide a formalism for a compact representation and evaluation of such scenarios.
- More complex semantics, especially in combination with an increasing amount of data, requires an automated computation of such solutions.
- Most of these problems are intractable, so implementing dedicated systems from the scratch is not the best idea.
- Distinction between direct implementation and reduction-based approach.
- We focus on reductions to propositional logic and Answer-Set Programming (ASP).
Outline

1 Direct- vs. Reduction-based Approach
2 Answer-Set Programming
3 ASP Approach to Abstract Argumentation
Laziness and Implementations

Alternative 1: The eastern way

- Implement a separate algorithm for each reasoning task
- Implementation is complicated because most reasoning tasks are inherently intricate (the complexity results given before)
- Implementation, testing, etc. require much effort and time
Laziness and Implementations

Alternative 1: The eastern way
- Implement a separate algorithm for each reasoning task
- Implementation is complicated because most reasoning tasks are inherently intricate (the complexity results given before)
- Implementation, testing, etc. require much effort and time

Alternative 2: The southern way
- Life is short; try to keep your effort as small as possible
- Let others work for you and use their results and software
- Be smart; apply what you have learned
The rapid implementation approach (RIA)

We know:

- Any complete problem can be translated into any other complete problem of the same complexity class
- Moreover, there exists poly-time translations (reductions)
- Complexity results (incl. completeness) for many reasoning tasks

We used already:

- e.g., the PTIME reduction from a CNF $\varphi$ to an AF $F(\varphi)$ such that $\varphi$ is satisfiable iff $F(\varphi)$ has an admissible set containing $\varphi$
- Can we “reverse” the reduction, i.e., from AFs to formulas?
- YES! Reduce to formalisms for which “good” solvers are available
  - But we have to find the PTIME reduction!
The rapid implementation approach (2)

- Reduce reasoning tasks for AF, e.g., to SAT problems of (Q)BFs
- Reductions are “cheap” (wrt. runtime and implementation effort!)
- Good SAT and QSAT solvers are available; simply use them

Benefits:

- Reductions are much easier to implement than full-fledged algorithms especially for “hard” reasoning tasks
- Basic reductions can be combined and reused
- Different formalisms can be reduced to same target formalism → beneficial for comparative studies
The rapid implementation approach (3)

Target formalisms are:

- The SAT problem for propositional formulas
- The SAT problem for quantified Boolean formulas
- Answer-set programs

Tools are available to solve all these three formalisms
Many developers are happy to give away their tool
They work hard to improve the tool’s performance (for you!)
Required properties of reductions: Faithfulness

- Let $\Pi$ be a decision problem
- $F_\Pi(\cdot)$ a reduction to a target formalism
- $F_\Pi(\cdot)$ has to satisfy the following three conditions:
  1. $F_\Pi(\cdot)$ is faithful, i.e., $F_\Pi(K)$ is true iff $K$ is a yes-instance of $\Pi$
  2. For each instance $K$, $F_\Pi(K)$ is poly-time computable wrt size of $K$
  3. Determining the truth of $F_\Pi(K)$ is computationally not harder than deciding $\Pi$

Faithfulness guarantees a correct “simulation” of $K$
Outline

1. Direct- vs. Reduction-based Approach
2. Answer-Set Programming
3. ASP Approach to Abstract Argumentation
General Idea of Answer-Set Programming

Fundamental concept:

- **Models** = set of atoms
- Models, not proofs, represent solutions!
- Need techniques to **compute models** (not to compute proofs)

Methodology to solve **search problems**

Solving search problems with ASP

- Given a problem $\Pi$ and an instance $K$, reduce it to the problem of computing intended models of a logic program:
  1. Encode $(\Pi, K)$ as a logic program $P$ such that the solutions of $\Pi$ for the instance $K$ are represented by the intended models of $P$
  2. Compute one intended model $M$ (an "answer set") of $P$
  3. Reconstruct a solution for $K$ from $M$

- Variant: Compute all intended models to obtain all solutions
Efficient solvers available

- gringo/clasp, clingo (University of Potsdam)
- dlv (TU Wien, University of Calabria)
- smodels, GnT (Aalto University, Finland)
- ASSAT (Hong Kong University of Science and Technology)
Answer-Set Programming Syntax

- We assume a first-order vocabulary $\Sigma$ comprised of nonempty finite sets of constants, variables, and predicate symbols, but no function symbols.
- A term is either a variable or a constant.
- An atom is an expression of form $p(t_1, \ldots, t_n)$, where
  - $p$ is a predicate symbol of arity $n \geq 0$ from $\Sigma$, and
  - $t_1, \ldots, t_n$ are terms.
- A literal is an atom $p$ or a negated atom $\neg p$.
  - $\neg$ is called strong negation, or classical negation.
- A literal is ground if it contains no variable.
A rule $r$ is an expression of the form

$$a_1 \lor \cdots \lor a_n \leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m,$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and “not” stands for default negation.

We call

- $H(r) = \{a_1, \ldots, a_n\}$ the **head** of $r$;
- $B(r) = \{b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m\}$ the **body** of $r$;
- $B^+(r) = \{b_1, \ldots, b_k\}$ the **positive body** of $r$;
- $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ the **negative body** of $r$.

Intuitive meaning of $r$: if $b_1, \ldots, b_k$ are derivable, but $b_{k+1}, \ldots, b_m$ are not derivable, then one of $a_1, \ldots, a_n$ is asserted.

- A **program** is a finite set of rules.
A rule $a_1 \lor \cdots \lor a_n \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m$ is

- a fact if $m = 0$ and $n \geq 1$
- a constraint if $n = 0$ (i.e., the head is empty)
- basic if $m = k$ and $n \geq 1$
- non-disjunctive if $n = 1$
- normal if it is non-disjunctive and contains no strong negation $\neg$
- Horn if it is normal and basic
- ground if all its literals are ground

A program is basic, normal, etc., if all of its rules are
ASP Semantics

- An interpretation $I$ satisfies a ground rule $r$ iff $H(r) \cap I \neq \emptyset$ whenever
  - $B^+(r) \subseteq I$,
  - $B^-(r) \cap I = \emptyset$.
- $I$ satisfies a ground program $\pi$, if each $r \in \pi$ is satisfied by $I$.
- A non-ground rule $r$ (resp., a program $\pi$) is satisfied by an interpretation $I$ iff $I$ satisfies all groundings of $r$ (resp., $Gr(\pi)$).

**Gelfond-Lifschitz reduct**

An interpretation $I$ is an answer set of $\pi$ iff it is a subset-minimal set satisfying

$$\pi^I = \{H(r) \leftarrow B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\pi)\}.$$
Outline

1 Direct- vs. Reduction-based Approach

2 Answer-Set Programming
   Guess and Check Methodology

3 ASP Approach to Abstract Argumentation
   ASP Encodings for Argumentation Semantics
   Saturation Encodings for Preferred
   Optimized Encodings for Preferred
Programming Methodology

Simplest technique: Guess and check

- **Guess**: Generate candidates for answer sets in the first step
- **Check**: Filter the answer sets and delete undesirable ones

Example (Graph coloring)

```prolog
node(a). node(b). node(c). edge(a, b). edge(b, c).  \} facts
col(red, X) \lor col(green, X) \lor col(blue, X) ← node(X). \} guess
← edge(X, Y), col(C, X), col(C, Y). \} check

G: Generate all possible coloring candidates
C: Delete all candidates where adjacent nodes have same color
```
# Complexity of Argumentation

<table>
<thead>
<tr>
<th></th>
<th>adm</th>
<th>pref</th>
<th>semi</th>
<th>stage</th>
<th>grd*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cred</td>
<td>NP-c</td>
<td>NP-c</td>
<td>(\Sigma^p_2)-c</td>
<td>(\Sigma^p_2)-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>Skept</td>
<td>(trivial)</td>
<td>(\Pi^p_2)-c</td>
<td>(\Pi^p_2)-c</td>
<td>(\Pi^p_2)-c</td>
<td>co-NP-c</td>
</tr>
</tbody>
</table>

[Baroni et al. 11; Dimopoulos & Torres 96; Dunne & Bench-Capon 02; Dvořák & Woltran 10]

---

# Recall: Data-Complexity of Datalog

<table>
<thead>
<tr>
<th></th>
<th>normal programs</th>
<th>disjunctive program</th>
<th>optimization programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[c]</td>
<td>NP</td>
<td>(\Sigma^p_2)</td>
<td>(\Sigma^p_2)</td>
</tr>
<tr>
<td>[s]</td>
<td>co-NP</td>
<td>(\Pi^p_2)</td>
<td>(\Pi^p_2)</td>
</tr>
</tbody>
</table>

[Dantsin,Eiter,Gottlob,Voronkov 01]
Outline

1 Direct- vs. Reduction-based Approach
2 Answer-Set Programming
3 ASP Approach to Abstract Argumentation
Conflict-free Set

Given an AF \((A, R)\).
A set \(S \subseteq A\) is conflict-free in \(F\), if, for each \(a, b \in S\), \((a, b) \notin R\).

Encoding for \(F = (A, R)\)

\[
\hat{F} = \{\text{arg}(a) \mid a \in A\} \cup \{\text{att}(a, b) \mid (a, b) \in R\}
\]

\[
\pi_{cf} = \begin{cases} 
\text{in}(X) & \leftarrow \text{not out}(X), \text{arg}(X) \\
\text{out}(X) & \leftarrow \text{not in}(X), \text{arg}(X) \\
 & \leftarrow \text{in}(X), \text{in}(Y), \text{att}(X, Y)
\end{cases}
\]

Result: For each AF \(F\), \(cf(F) \equiv \mathcal{A}S(\pi_{cf}(\hat{F}))\)
Outline

1. Direct- vs. Reduction-based Approach

2. Answer-Set Programming
   Guess and Check Methodology

3. ASP Approach to Abstract Argumentation
   ASP Encodings for Argumentation Semantics
   Saturation Encodings for Preferred
   Optimized Encodings for Preferred
Admissible Sets

Given an AF $F = (A, R)$. A set $S \subseteq A$ is admissible in $F$, if

- $S$ is conflict-free in $F$
- each $a \in S$ is defended by $S$ in $F$
- $a \in A$ is defended by $S$ in $F$, if for each $b \in A$ with $(b, a) \in R$, there exists a $c \in S$, such that $(c, b) \in R$.

Encoding

$$\pi_{adm} = \pi_{cf} \cup \{ \text{defeated}(X) \leftarrow \text{in}(Y), \text{att}(Y, X) \leftarrow \text{in}(X), \text{att}(Y, X), \text{not defeated}(Y) \}$$

Result: For each AF $F$, $adm(F) \equiv AS(\pi_{adm}(\hat{F}))$
Stable Extensions

Given an AF $F = (A, R)$. A set $S \subseteq A$ is a stable extension of $F$, if

- $S$ is conflict-free in $F$
- for each $a \in A \setminus S$, there exists a $b \in S$, such that $(b, a) \in R$

Encoding

$\pi_{stable} = \pi_{cf} \cup \{\text{defeated}(X) \leftarrow \text{in}(Y), \text{att}(Y, X) \leftarrow \text{out}(X), \text{not defeated}(X)\}$

Result: For each AF $F$, $\text{stable}(F) \equiv \text{AS}(\pi_{stable}(\hat{F}))$
Grounded Extension

Given an AF $F = (A, R)$. The characteristic function $\mathcal{F}_F : 2^A \rightarrow 2^A$ of $F$ is defined as

$$\mathcal{F}_F(E) = \{x \in A \mid x \text{ is defended by } E\}.$$ 

The least fixed point of $\mathcal{F}_F$ is the grounded extension.

Order over domain

$$\pi< = \begin{cases} 
\text{lt}(X, Y) & \leftarrow \text{arg}(X), \text{arg}(Y), X < Y \\
\text{nsucc}(X, Z) & \leftarrow \text{lt}(X, Y), \text{lt}(Y, Z) \\
\text{succ}(X, Y) & \leftarrow \text{lt}(X, Y), \text{not nsucc}(X, Y) \\
\text{ninf}(X) & \leftarrow \text{lt}(Y, X) \\
\text{nsup}(X) & \leftarrow \text{lt}(X, Y) \\
\text{inf}(X) & \leftarrow \text{not ninf}(X), \text{arg}(X) \\
\text{sup}(X) & \leftarrow \text{not nsup}(X), \text{arg}(X) 
\end{cases}$$
**Grounded Extension**

Given an AF $F = (A, R)$. The characteristic function $F_F : 2^A \rightarrow 2^A$ of $F$ is defined as

$$F_F(E) = \{ x \in A \mid x \text{ is defended by } E \}.$$ 

The least fixed point of $F_F$ is the grounded extension.

**Encodings Grounded Extension**

$$\pi_{ground} = \left\{ \begin{array}{ll}
\text{def}_\text{upto}(X, Y) & \leftarrow \text{inf}(Y), \text{arg}(X), \text{not att}(Y, X) \\
\text{def}_\text{upto}(X, Y) & \leftarrow \text{inf}(Y), \text{in}(Z), \text{att}(Z, Y), \text{att}(Y, X) \\
\text{def}_\text{upto}(X, Y) & \leftarrow \text{succ}(Z, Y), \text{def}_\text{upto}(X, Z), \text{not att}(Y, X) \\
\text{def}_\text{upto}(X, Y) & \leftarrow \text{succ}(Z, Y), \text{def}_\text{upto}(X, Z), \text{in}(V), \text{att}(V, Y), \text{att}(Y, X) \\
\text{defended}(X) & \leftarrow \text{sup}(Y), \text{def}_\text{upto}(X, Y) \\
\text{in}(X) & \leftarrow \text{defended}(X) \\
\end{array} \right\}$$

**Result:** For each AF $F$, $\text{ground}(F) \equiv \mathcal{AS}(\pi_{\text{ground}}(\hat{F}))$
Preferred Extensions

Given an AF $F = (A, R)$. A set $S \subseteq A$ is a preferred extension of $F$, if

- $S$ is admissible in $F$
- for each $T \subseteq A$ admissible in $F$, $S \not\subset T$

Encoding

- Preferred semantics needs subset maximization task.
- Can be encoded in standard ASP but requires insight and expertise.
Outline

1 Direct- vs. Reduction-based Approach

2 Answer-Set Programming
   Guess and Check Methodology

3 ASP Approach to Abstract Argumentation
   ASP Encodings for Argumentation Semantics
   Saturation Encodings for Preferred
   Optimized Encodings for Preferred
Saturation Encodings

Preferred Extension
Given an AF \((A, R)\). A set \(S \subseteq A\) is preferred in \(F\), if \(S\) is admissible in \(F\) and for each \(T \subseteq A\) admissible in \(T\), \(S \not\subset T\).

Encoding
\[
\pi_{\text{saturate}} = \begin{cases}
\text{inN}(X) \lor \text{outN}(X) & \leftarrow \text{out}(X); \\
\text{inN}(X) & \leftarrow \text{in}(X) \\
\text{spoil} & \leftarrow \text{eq} \\
\text{spoil} & \leftarrow \text{inN}(X), \text{inN}(Y), \text{att}(X, Y) \\
\text{spoil} & \leftarrow \text{inN}(X), \text{outN}(Y), \text{att}(Y, X), \text{undefeated}(Y) \\
\text{inN}(X) & \leftarrow \text{spoil}, \text{arg}(X) \\
\text{outN}(X) & \leftarrow \text{spoil}, \text{arg}(X) \\
& \leftarrow \text{not spoil}
\end{cases}
\]

\[
\pi_{\text{pref}} = \pi_{\text{adm}} \cup \pi_{\text{helpers}} \cup \pi_{\text{saturate}}
\]

Result: For each AF \(F\), \(\text{pref}(F) \equiv \text{AS}(\pi_{\text{pref}}(\hat{F}))\)
Check if second guess is equal to the first one.

<table>
<thead>
<tr>
<th>equpto(Y)</th>
<th>←</th>
<th>inf(Y), in(Y), inN(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>equpto(Y)</td>
<td>←</td>
<td>inf(Y), out(Y), outN(Y)</td>
</tr>
<tr>
<td>equpto(Y)</td>
<td>←</td>
<td>succ(Z, Y), in(Y), inN(Y), equpto(Z)</td>
</tr>
<tr>
<td>equpto(Y)</td>
<td>←</td>
<td>succ(Z, Y), out(Y), outN(Y), equpto(Z)</td>
</tr>
<tr>
<td>eq</td>
<td>←</td>
<td>sup(Y), equpto(Y)</td>
</tr>
</tbody>
</table>
Outline

1 Direct- vs. Reduction-based Approach

2 Answer-Set Programming
   Guess and Check Methodology

3 ASP Approach to Abstract Argumentation
   ASP Encodings for Argumentation Semantics
   Saturation Encodings for Preferred
   Optimized Encodings for Preferred
Alternative Characterization for Preferred
[Gaggl et al., 2015]

Proposition 1

Let $F = (A, R)$ be an AF and $S \subseteq A$ be admissible in $F$. Then, $S \in \operatorname{pref}(F)$ iff, for each $E \in \operatorname{adm}(F)$ such that $E \not\subseteq S$, $E \cup S \not\in \operatorname{cf}(F)$.

Example

$\operatorname{adm}(F) = \{\emptyset, \{a\}, \{c\}, \{a, c\}, \{a, d\}, \{c, f\}, \{a, c, f\}, \{a, d, f\}\}$, and

$\operatorname{pref}(F) = \{\{a, c, f\}, \{a, d, f\}\}$

TU Dresden, ICCL Summer School 2017 Abstract Argumentation slide 33 of 49
New Encodings for Preferred

**Proposition 1**

Let \( F = (A, R) \) be an AF and \( S \subseteq A \) be admissible in \( F \). Then, \( S \in \text{pref}(F) \) iff, for each \( E \in \text{adm}(F) \) such that \( E \nsubseteq S, E \cup S \notin \text{cf}(F) \).

\[
\pi_{\text{satpref}}^2 = \begin{cases} 
\text{nontrivial} & \leftarrow \text{out}(X) \\
\text{witness}(X) : \text{out}(X) & \leftarrow \text{nontrivial} \\
\text{spoil|witness}(Z) : \text{att}(Z, Y) & \leftarrow \text{witness}(X), \text{att}(Y, X) \\
\text{spoil} & \leftarrow \text{att}(X, Y), \text{witness}(X), \text{witness}(Y) \\
\text{spoil} & \leftarrow \text{in}(X), \text{witness}(Y), \text{att}(X, Y) \\
\text{witness}(X) & \leftarrow \text{spoil, arg}(X) \\
\end{cases}
\]

\[
\pi_{\text{pref}}^2 = \pi_{\text{adm}} \cup \pi_{\text{satpref}}^2
\]

**Result:** For each AF \( F \), \( \text{pref}(F) \equiv \mathcal{A}S(\pi_{\text{pref}}^2(\hat{F})) \)
Functionality of New Encodings

\[
\text{nontrivial} \quad \leftarrow \quad \text{out}(X) \\
\text{witness}(X) : \text{out}(X) \quad \leftarrow \quad \text{nontrivial}
\]

Example

\[
\begin{array}{c}
a \rightarrow b \\
c \rightarrow d \\
d \rightarrow e \\
e \rightarrow f
\end{array}
\]

TU Dresden, ICCL Summer School 2017 Abstract Argumentation
Functionality of New Encodings

nontrivial ← out(X)
witness(X) : out(X) ← nontrivial

Example
nontrivial ← out(X)
witness(X) : out(X) ← nontrivial
spoil|witness(Z) : att(Z, Y) ← witness(X), att(Y, X)

Example
nontrivial ← out(X)
witness(X) : out(X) ← nontrivial
spoil|witness(Z) : att(Z, Y) ← witness(X), att(Y, X)
spoil ← att(X, Y), witness(X), witness(Y)

Example
Functionality of New Encodings

\[
\begin{align*}
\text{nontrivial} & \quad \leftarrow \text{out}(X) \\
\text{witness}(X) : \text{out}(X) & \quad \leftarrow \text{nontrivial} \\
\text{spoil} | \text{witness}(Z) : \text{att}(Z, Y) & \quad \leftarrow \text{witness}(X), \text{att}(Y, X) \\
\text{spoil} & \quad \leftarrow \text{att}(X, Y), \text{witness}(X), \text{witness}(Y) \\
\text{spoil} & \quad \leftarrow \text{in}(X), \text{witness}(Y), \text{att}(X, Y)
\end{align*}
\]

Example

TU Dresden, ICCL Summer School 2017 Abstract Argumentation slide 39 of 49
Functionality of New Encodings

nontrivial ← out(X)

witness(X) : out(X) ← nontrivial

spoil | witness(Z) : att(Z, Y) ← witness(X), att(Y, X)

spoil ← att(X, Y), witness(X), witness(Y)

spoil ← in(X), witness(Y), att(X, Y)

witness(X) ← spoil, arg(X)

← not spoil, nontrivial

Example

TU Dresden, ICCL Summer School 2017 Abstract Argumentation
Proposition 1

Let $F = (A, R)$ be an AF and $S \subseteq A$ be admissible in $F$. Then, $S \in \text{pref}(F)$ iff, for each $E \in \text{adm}(F)$ such that $E \not\subseteq S$, $E \cup S \not\in \text{cf}(F)$.

Example
Positive Example

Example

\[
\begin{align*}
\text{nontrivial} & \leftarrow \text{out}(X) \\
\text{witness}(X) : \text{out}(X) & \leftarrow \text{nontrivial}
\end{align*}
\]
Positive Example

nontrivial ← out(X)
witness(X) : out(X) ← nontrivial

Example
Positive Example

\[
\begin{align*}
\text{nontrivial}
& \quad \leftarrow \quad \text{out}(X) \\
\text{witness}(X) : \text{out}(X)
& \quad \leftarrow \quad \text{nontrivial} \\
\text{spoil} | \text{witness}(Z) : \text{att}(Z, Y)
& \quad \leftarrow \quad \text{witness}(X), \text{att}(Y, X)
\end{align*}
\]

Example
### Positive Example

| nontrivial | ← out($X$) |
| witness($X$) : out($X$) | ← nontrivial |
| spoil | ← witness($X$), att($Y$, $X$) |
| witness($Z$) : att($Z$, $Y$) | ← att($X$, $Y$), witness($X$), witness($Y$) |

#### Example

![Graph Diagram]

- $a$ to $b$
- $b$ to $a$
- $b$ to $c$
- $c$ to $b$
- $c$ to $d$
- $d$ to $e$
- $e$ to $f$
- $f$ to $e$

TU Dresden, ICCL Summer School 2017 Abstract Argumentation
Positive Example

Example

nontrivial \leftarrow \text{out}(X)
\text{witness}(X) : \text{out}(X) \leftarrow \text{nontrivial}
\text{spoil} \mid \text{witness}(Z) : \text{att}(Z, Y) \leftarrow \text{witness}(X), \text{att}(Y, X)
\text{spoil} \leftarrow \text{att}(X, Y), \text{witness}(X), \text{witness}(Y)
\text{spoil} \leftarrow \text{in}(X), \text{witness}(Y), \text{att}(X, Y)
### Positive Example

<table>
<thead>
<tr>
<th>nontrivial</th>
<th>←</th>
<th>out(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>witness(X) : out(X)</td>
<td>←</td>
<td>nontrivial</td>
</tr>
<tr>
<td>spoil</td>
<td>witness(Z) : att(Z, Y)</td>
<td>←</td>
</tr>
<tr>
<td>spoil</td>
<td>←</td>
<td>att(X, Y), witness(X), witness(Y)</td>
</tr>
<tr>
<td>spoil</td>
<td>←</td>
<td>in(X), witness(Y), att(X, Y)</td>
</tr>
<tr>
<td>witness(X)</td>
<td>←</td>
<td>spoil, arg(X)</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>not spoil, nontrivial</td>
</tr>
</tbody>
</table>

### Example

```
\begin{tikzpicture}[auto, node distance = 2cm, >=latex]
    \node (a) at (0,0) {$a$};
    \node (b) at (1,1) {$b$};
    \node (c) at (2,2) {$c$};
    \node (d) at (1,0) {$d$};
    \node (e) at (2,-1) {$e$};
    \node (f) at (3,0) {$f$};

    \path[->] (a) edge (b);
    \path[->] (b) edge (c);
    \path[->] (b) edge (d);
    \path[->] (c) edge (d);
    \path[->] (d) edge (e);
    \path[->] (e) edge (f);
\end{tikzpicture}
```
Philippe Besnard and Sylvie Doutre.
Checking the acceptability of a set of arguments.

S. Bistarelli, F. Santini, Conarg: a tool to solve (weighted) abstract argumentation frameworks with (soft) constraints, CoRR abs/1212.2857.

On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.

Making use of advances in answer-set programming for abstract argumentation systems.

Uwe Egly and Stefan Woltran.
Reasoning in argumentation frameworks using quantified boolean formulas.

Uwe Egly, Sarah Gaggl, and Stefan Woltran.
Answer-set programming encodings for argumentation frameworks.

Sarah Alice Gaggl, Norbert Manthey, Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran
Improved answer-set programming encodings for abstract argumentation.

Complex optimization in answer set programming.