

# Utilizing ASP for Generating and Visualizing Argumentation Frameworks

Günther Charwat   Johannes Peter Wallner   Stefan Woltran

Database and Artificial Intelligence Group  
Institute of Information Systems  
Vienna University of Technology

ASPOCP'12 - September 4, 2012

# Outline

## 1. Argumentation

### 1.1 Argumentation Frameworks

### 1.2 Our Motivation

## 2. ASP Encodings

### 2.1 Overall Approach

### 2.2 Model Checking

### 2.3 Forming Arguments

### 2.4 Identifying Conflicts between Arguments

## 3. Visualization of Argumentation Frameworks

## 4. Summary and Future Work

# Argumentation - Argumentation Frameworks

## Steps

Starting point:  
knowledge-base

- 1) Form arguments
- 2) Identify conflicts
- 3) Abstract from  
internal structure
- 4) Resolve conflicts
- 5) Obtain conclusions

# Argumentation - Argumentation Frameworks

## Steps

### Starting point: knowledge-base

- 1) Form arguments
- 2) Identify conflicts
- 3) Abstract from internal structure
- 4) Resolve conflicts
- 5) Obtain conclusions

- Input: A knowledge-base  $\mathcal{K}$  and set  $\mathcal{C}$  of claims

## Example

$$\mathcal{K} = \{a, a \rightarrow b, \neg b\}$$
$$\mathcal{C} = \mathcal{K} \cup \{\neg a, b, a \wedge \neg b\}$$

# Argumentation - Argumentation Frameworks

## Steps

Starting point:  
knowledge-base

- 1) **Form arguments**
- 2) Identify conflicts
- 3) Abstract from internal structure
- 4) Resolve conflicts
- 5) Obtain conclusions

- Form arguments  $A = (S, C)$  consisting of support  $S \subseteq \mathcal{K}$  and claim  $C \in \mathcal{C}$

## Example

$$\mathcal{K} = \{a, a \rightarrow b, \neg b\}$$

$$\mathcal{C} = \mathcal{K} \cup \{\neg a, b, a \wedge \neg b\}$$

$$(\{a\}, a)$$

$$(\{\neg b, a \rightarrow b\}, \neg a)$$

$$(\{a, a \rightarrow b\}, b)$$

$$(\{a, \neg b\}, a \wedge \neg b)$$

$$(\{\neg b\}, \neg b)$$

$$(\{a \rightarrow b\}, a \rightarrow b)$$

# Argumentation - Argumentation Frameworks

## Steps

Starting point:  
knowledge-base

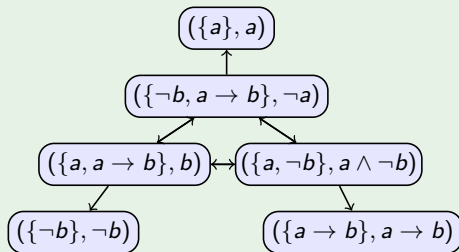
- 1) Form arguments
- 2) Identify conflicts**
- 3) Abstract from internal structure
- 4) Resolve conflicts
- 5) Obtain conclusions

- Identify conflicts between arguments  
 $A = (S, C)$  and  $A' = (S', C')$

## Example

$$\mathcal{K} = \{a, a \rightarrow b, \neg b\}$$

$$\mathcal{C} = \mathcal{K} \cup \{\neg a, b, a \wedge \neg b\}$$



# Argumentation - Argumentation Frameworks

## Steps

Starting point:  
knowledge-base

- 1) Form arguments
- 2) Identify conflicts
- 3) **Abstract from internal structure**
- 4) Resolve conflicts
- 5) Obtain conclusions

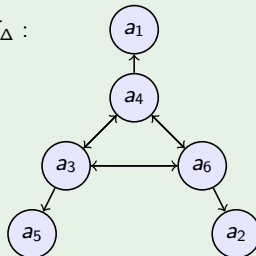
- Obtain an Abstract Argumentation Framework  $F_{\Delta}$  (Dung [1995])

## Example

$$\mathcal{K} = \{a, a \rightarrow b, \neg b\}$$

$$\mathcal{C} = \mathcal{K} \cup \{\neg a, b, a \wedge \neg b\}$$

$F_{\Delta}$  :



# Argumentation - Argumentation Frameworks

## Steps

Starting point:  
knowledge-base

- 1) Form arguments
- 2) Identify conflicts
- 3) Abstract from internal structure
- 4) **Resolve conflicts**
- 5) Obtain conclusions

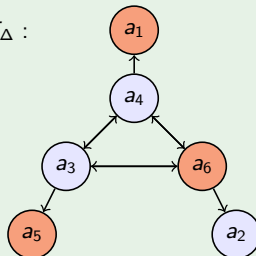
- Based on a semantics, select arguments that 'can stand together'

## Example

$$\mathcal{K} = \{a, a \rightarrow b, \neg b\}$$

$$\mathcal{C} = \mathcal{K} \cup \{\neg a, b, a \wedge \neg b\}$$

$F_{\Delta}$  :



$$stb(F_{\Delta}) = \{ \{a_1, a_5, a_6\}, \{a_1, a_2, a_3\}, \{a_2, a_4, a_5\} \}$$



# Argumentation - Argumentation Frameworks

## Steps

Starting point:  
knowledge-base

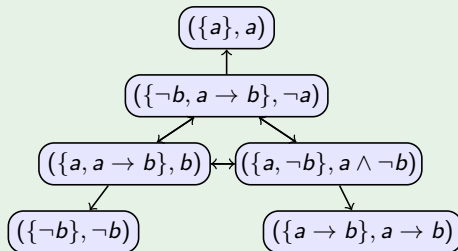
- 1) Form arguments
- 2) Identify conflicts
- 3) Abstract from internal structure
- 4) Resolve conflicts
- 5) **Obtain conclusions**

- Derive deductive closure of contents from accepted arguments

## Example

$$\mathcal{K} = \{a, a \rightarrow b, \neg b\}$$

$$\mathcal{C} = \mathcal{K} \cup \{\neg a, b, a \wedge \neg b\}$$



$$Cn_{stb}(F_{\Delta}) = Cn((a \wedge \neg b) \vee (a \wedge b \wedge a \rightarrow b) \vee (a \rightarrow b \wedge \neg a \wedge \neg b))$$

# Argumentation - Our Motivation

## Steps

Starting point:  
knowledge-base

- 1) **Form arguments**
- 2) **Identify conflicts**
- 3) Abstract from  
internal structure
- 4) Resolve conflicts
- 5) Obtain conclusions

## Current Status

- **Not many** systems for **instantiation of AFs**:  
e.g. CARNEADES, TOAST (for ASPIC+)
- **Many** different systems for **evaluating AFs**:  
e.g. ASPARTIX, Dungere, CASAPI,  
ArguMed, ConArg, reasoner by Visser (2008)
- **Few** tools for **combining Computation and Visualization**:  
e.g. CARNEADES, ASPARTIX (Web)

# Argumentation - Our Motivation

## Steps

Starting point:  
knowledge-base

- 1) Form arguments
- 2) Identify conflicts
- 3) Abstract from internal structure
- 4) **Resolve conflicts**
- 5) Obtain conclusions

## Current Status

- **Not many** systems for **instantiation of AFs**:  
e.g. CARNEADES, TOAST (for ASPIC+)
- **Many** different systems for **evaluating AFs**:  
e.g. ASPARTIX, Dungine, CASAPI, ArguMed, ConArg, reasoner by Visser (2008)
- **Few** tools for **combining Computation and Visualization**:  
e.g. CARNEADES, ASPARTIX (Web)

# Argumentation - Our Motivation

## Steps

Starting point:  
knowledge-base

- 1) Form arguments
- 2) Identify conflicts
- 3) Abstract from internal structure
- 4) Resolve conflicts
- 5) Obtain conclusions

## Current Status

- **Not many** systems for **instantiation of AFs**:  
e.g. CARNEADES, TOAST (for ASPIC+)
- **Many** different systems for **evaluating AFs**:  
e.g. ASPARTIX, Dungine, CASAPI, ArguMed, ConArg, reasoner by Visser (2008)
- **Few** tools for **combining Computation and Visualization**:  
e.g. CARNEADES, ASPARTIX (Web)

# Argumentation - Our Motivation

## Steps

Starting point:  
knowledge-base

- 1) **Form arguments**
- 2) **Identify conflicts**
- 3) **Abstract from internal structure**
- 4) Resolve conflicts
- 5) Obtain conclusions

## Current Status

- **Not many** systems for **instantiation of AFs**:  
e.g. CARNEADES, TOAST (for ASPIC+)
- **Many** different systems for **evaluating AFs**:  
e.g. ASPARTIX, Dungine, CASAPI, ArguMed, ConArg, reasoner by Visser (2008)
- **Few** tools for **combining Computation and Visualization**:  
e.g. CARNEADES, ASPARTIX (Web)

## Our Goals

- **Instantiation** of AFs via ASP encodings
- Tool for handling instantiation process and **visualization** of obtained Argumentation Frameworks

# ASP Encodings

Overall Approach (Besnard and Hunter [2001]):

1 Form **arguments**  $A = (S, C)$  such that for each argument:

- 1 Input:  $S \subseteq \mathcal{K}$  and  $C \in \mathcal{C}$
- 2 **Consistency**:  $S$  consistent
- 3 **Entailment**:  $S \models C$
- 4 **Subset Minimality**:  $\nexists S' \subset S$  s.t.  $S' \models C$

2 Identify **conflicts** between arguments  $A = (S, C)$  and  $A' = (S', C')$ :

- Variety of different attack types, expressed by satisfiability of formulae, e.g.:
- **Defeat**:  $C \models \neg S'$
- **Direct Defeat**:  $C \models \neg\phi'_i$  for a  $\phi'_i \in S'$
- **Rebuttal**:  $C \equiv \neg C'$
- ...

→ **Model checking** via ASP basis for construction of AFs

→ **Two-step** approach: One encoding for arguments, another for attacks

# Model Checking

- Basis for construction of arguments and attack relations
- Any **propositional logic** formula allowed (e.g.  $\text{imp}(a, \text{neg}(b))$ )

Model Checking Encoding ( $\pi_{\text{modelcheck}}$ ):

- 1) Input: Formula, specified by  $\text{formula}(F)$
- 2) Splitting of formula in subformulae and contained atoms
- 3) Guess interpretations, e.g.  $\text{true}(k, A) \vee \text{false}(k, A) \leftarrow \text{atom}(A)$ .
- 4) Obtain either  $\text{ismodel}(k, F)$  or  $\text{nomodel}(k, F)$  for formula  $F$

## Example

Input:  $\text{formula}(\text{imp}(a, \text{neg}(b)))$

Output (amongst others):  $\{\text{false}(k, a). \text{true}(k, b). \text{ismodel}(k, \text{imp}(a, \text{neg}(b)))\}$ .

# Forming Arguments

(1) Input:  $S \subseteq \mathcal{K}$  and  $C \in \mathcal{C}$

- Knowledge-base  $\mathcal{K}$ , represented by the predicate  $\text{kb}(\cdot)$
- A set  $\mathcal{C}$  of claims, represented by the predicate  $\text{cl}(\cdot)$

Guess arguments:

$$\pi_{\text{arg}} = \{ \begin{array}{l} 1\{ \text{claim}(C) : \text{cl}(C) \}1; \\ 1\{ \text{fs}(FS) : \text{kb}(FS) \}; \\ \text{formula}(C) \leftarrow \text{claim}(C); \\ \text{formula}(FS) \leftarrow \text{fs}(FS). \end{array} \}$$

(2) Consistency:  $S$  consistent

$$\pi_{\text{consistent}} = \{ \begin{array}{l} 1\{ \text{true}(\text{consistent}, A), \text{false}(\text{consistent}, A) \}1 \leftarrow \text{atom}(A). \\ \leftarrow \text{nomodel}(\text{consistent}, FS), \text{fs}(FS). \end{array} \}$$



## Forming Arguments

### (3) Entailment: $S \models C$

- Expressible by unsatisfiability of  $\neg(S \rightarrow C) \equiv \neg(\neg S \vee C) \equiv S \wedge \neg C$
- We apply the **saturation technique** (Eiter and Gottlob [1995])

```

 $\pi_{\text{entailment}} = \{$ 
  true(entail, A)  $\vee$  false(entail, A)  $\leftarrow$  atom(A);
  entails_claim  $\leftarrow$  nomodel(entail, neg(C)), claim(C);
  entails_claim  $\leftarrow$  nomodel(entail, FS), fs(FS);
  true(entail, A)  $\leftarrow$  entails_claim, atom(A);
  false(entail, A)  $\leftarrow$  entails_claim, atom(A) :
   $\leftarrow$  not entails_claim.  $\}$ 

```

- If  $S$  or  $C$  not satisfied for interpretation, we obtain entails\_claim and saturate
- Otherwise, constraint  $\leftarrow$  not entails\_claim removes answer set
- Due to stable model semantics, answer set is only returned in case  $S \wedge \neg C$  is unsatisfiable

# Forming Arguments

(4) Subset Minimality:  $\nexists S' \subset S$  s.t.  $S' \models C$

- We apply concept of **loop** (see e.g. Eiter et al. [2009])
- All  $S' \subset S$  considered where exactly one formula  $\alpha \in S$  but  $\alpha \notin S'$
- Sufficient due to monotonicity of classical logic

Minimality Encoding ( $\pi_{\text{minimize}}$ ):

- For each  $S'$ , check if  $S' \models C$  valid
- **Only** keep answer sets, where guessed interpretation is **no model** for  $S' \models C$
- If  $S' \models C$  valid,  $S$  is not subset minimal
  - All answer sets containing  $S$  as support are removed

# Forming Arguments

Arguments obtained by:

$$\pi_{\text{arguments}} = \pi_{\text{modelcheck}} \cup \pi_{\text{arg}} \cup \pi_{\text{consistent}} \cup \pi_{\text{entailment}} \cup \pi_{\text{minimize}}$$

## Example

Input:

$$\{ \text{kb}(a). \text{kb}(\text{imp}(a, b)). \text{kb}(\text{neg}(b)). \\ \text{cl}(a). \text{cl}(\text{imp}(a, b)). \text{cl}(\text{neg}(b)). \text{cl}(\text{neg}(a)). \text{cl}(b). \text{cl}(\text{and}(a, \text{neg}(b))). \}$$

Output:

$$\begin{aligned} a_1 : & \{ \text{fs}(a). \text{claim}(a). \} \\ a_2 : & \{ \text{fs}(\text{imp}(a, b)). \text{claim}(\text{imp}(a, b)). \} \\ a_3 : & \{ \text{fs}(a). \text{fs}(\text{imp}(a, b)). \text{claim}(b). \} \\ a_4 : & \{ \text{fs}(\text{neg}(b)). \text{fs}(\text{imp}(a, b)). \text{claim}(\text{neg}(a)). \} \\ a_5 : & \{ \text{fs}(\text{neg}(b)). \text{claim}(\text{neg}(b)). \} \\ a_6 : & \{ \text{fs}(a). \text{fs}(\text{neg}(b)). \text{claim}(\text{and}(a, \text{neg}(b))). \} \end{aligned}$$

# Identifying Conflicts between Arguments

## (1) Input: A set of arguments

- Consists of 'flattened' arguments obtained by  $\pi_{\text{arguments}}$
- Each argument specified by a set of predicates  $\text{as}(A, fs, \cdot)$  and the predicate  $\text{as}(A, claim, \cdot)$

## Example

$$\{ \text{as}(1, fs, a). \text{as}(1, claim, a). \text{as}(2, fs, \text{imp}(a, b)). \text{as}(2, claim, \text{imp}(a, b)). \text{as}(3, fs, a). \\ \text{as}(3, fs, \text{imp}(a, b)). \text{as}(3, claim, b). \}$$

## (2) Guess exactly two arguments ( $\pi_{\text{att}}$ )

## (3) Build single support formula ( $\pi_{\text{support}}$ )

- Define an ordering over support formulae  $\text{as}(A, fs, \cdot)$
- 'Iterate' over ordering and combine by conjunction

## Identifying Conflicts between Arguments

### (4) Define Attack Types (e.g. $\pi_{\text{defeat}}$ )

- Construct formula based on attack type  
e.g. Defeat:  $C \models \neg S'$  specified as  $\text{imp}(C, \text{neg}(S'))$
- Apply model checking to formula
- Derive predicate entails in case formula is satisfied

### (5) Saturate ( $\pi_{\text{att\_sat}}$ )

- Apply coNP check (similar to entailment for arguments)
- If entails not derived, answer set is removed
- Otherwise, we saturate
- If formula of **some** attack type valid, we saturate **all** attack types

Attacks (defeats) obtained by:

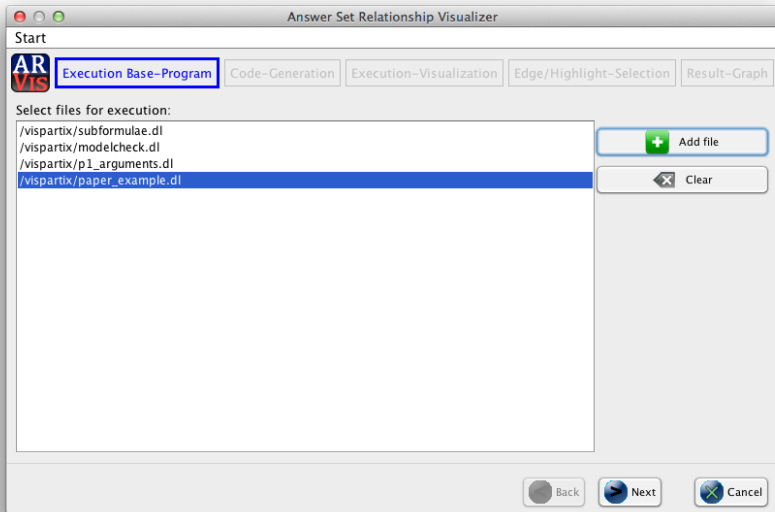
$$\pi_{\text{attacks}} = \pi_{\text{modelcheck}} \cup \pi_{\text{att}} \cup \pi_{\text{support}} \cup \pi_{\text{defeat}} \cup \pi_{\text{att\_sat}}$$

# Visualization of Argumentation Frameworks

We utilise the purpose built tool ARVis (Answer set Relationship Visualizer):

- 1 **Obtain arguments:** Provide  $\pi_{\text{arguments}}$  and a problem instance, gringo and claspD compute arguments
- 2 **Flatten arguments:** Generate argument facts
- 3 **Obtain attacks:** Provide  $\pi_{\text{attacks}}$  and any attack type programs, attacks are computed
- 4 **Argumentation Framework:** ARVis provides graph visualization consisting of arguments (vertices) and attacks (edges)
- 5 **Export:** Graph may be exported for further processing

# Visualization of Argumentation Frameworks



# Visualization of Argumentation Frameworks

The screenshot shows a window titled "Answer Set Relationship Visualizer" with a "Start" button. Below the title bar is a navigation bar with five buttons: "AR Vis" (with a logo), "Execution Base-Program", "Code-Generation" (highlighted with a blue border), "Execution-Visualization", "Edge/Highlight-Selection", and "Result-Graph".

The main area is labeled "Execution output:" and contains a list of six IDed lines of logic:

```
ID 1: fs(a) claim(a)
ID 2: fs(imp(a,b)) claim(imp(a,b))
ID 3: fs(imp(a,b)) fs(a) claim(b)
ID 4: fs(imp(a,b)) fs(neg(b)) claim(neg(a))
ID 5: fs(neg(b)) claim(neg(b))
ID 6: fs(neg(b)) fs(a) claim(and(a,neg(b)))
```

Below this is a section labeled "Select predicates for Answer-Set-Generation:" with a list box containing two entries: "claim / 1" and "fs / 1". The "fs / 1" entry is currently selected and highlighted in blue.

At the bottom right of the window are three buttons: "Back" (with a left arrow), "Next" (with a right arrow), and "Cancel" (with a close icon).



# Visualization of Argumentation Frameworks

Answer Set Relationship Visualizer

Start

AR Vis Execution Base-Program Code-Generation **Execution-Visualization** Edge/Highlight-Selection Result-Graph

Generated ASP-Code:

```

as(1, claim, a).
as(2, fs, imp(a,b)).
as(2, claim, imp(a,b)).
as(3, fs, imp(a,b)).
as(3, fs, a).
as(3, claim, b).
as(4, fs, imp(a,b)).
as(4, fs, neg(b)).
as(4, claim, neg(a)).
as(5, fs, neg(b)).
as(5, claim, neg(b)).
as(6, fs, neg(b)).
as(6, fs, a).
as(6, claim, and(a,neg(b))).
  
```

Choose constraints for visualization that have influence on the Graph:

```

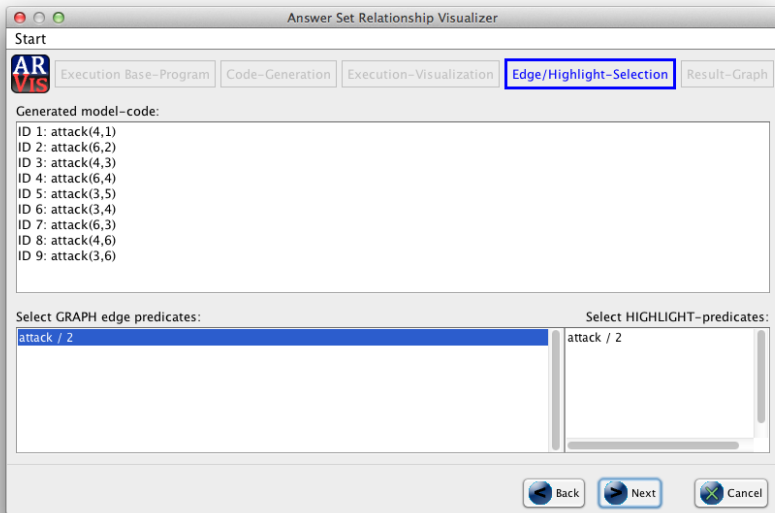
/vispartix/subformulae.dl
/vispartix/modelcheck.dl
/vispartix/p2_attacks.dl
/vispartix/p2_directdefeat.dl
  
```

+ Add visualization-file

X Clear

Back Next Cancel

# Visualization of Argumentation Frameworks



# Visualization of Argumentation Frameworks

Answer Set Relationship Visualizer

Start

AR Vis Execution Base-Program Code-Generation Execution-Visualization Edge/Highlight-Selection **Result-Graph**

Select a model:  
Model 1

Graph-Visualization:

```

graph TD
    1((1)) --> 4((4))
    4((4)) --> 3((3))
    4((4)) --> 6((6))
    3((3)) --> 5((5))
    6((6)) --> 2((2))
    3((3)) --> 6((6))
    6((6)) --> 3((3))
  
```

Mouse-Left: Select/Move one or more vertices  
Mouse-Wheel: Zoom  
Mouse-Right: Move complete Graph

Filter predicates:  
claim  
fs

Result of selected answersets (vertices):

AnswerSet 4:  
fs(imp(a,b)) fs(neg(b)) claim(neg(a))

AnswerSet 6:  
fs(neg(b)) fs(a) claim(and(a,neg(b)))

AnswerSet 5:  
fs(neg(b)) claim(neg(b))

Export txt-File

Back Finish Cancel

# Summary and Future Work

## Summary:





- ASP encodings for **instantiation of Argumentation Frameworks** (two steps)
  - Declaratively described
  - Easily adaptable to other notions
- New tool for **visualising relations between answer sets** (ARVis)
  - Not restricted to Argumentation domain
  - Versatile, e.g. export of graph for post-processing
- Encodings and tool publicly available:

`http://dbai.tuwien.ac.at/proj/argumentation/vispartix`

## Future Work:

- Implement further instantiation approaches
- Performance evaluation
- Analyze further application areas for answer set relationship visualization

# References

-  Besnard, P. and Hunter, A. (2001).  
A logic-based theory of deductive arguments.  
*Artif. Intell.*, 128(1-2):203–235.
-  Dung P. M. (1995).  
On the acceptability of arguments and its fundamental role in  
nonmonotonic reasoning, logic programming and n-person games.  
*Artif. Intell.*, 77(2):321–357.
-  Eiter, T. and Gottlob, G. (1995).  
On the Computational Cost of Disjunctive Logic Programming:  
Propositional Case.  
*Ann. Math. Artif. Intell.*, 15(3-4):289–323.
-  Eiter, T., Ianni, G., and Krennwallner, T. (2009).  
Answer Set Programming: A Primer.  
In *Reasoning Web*, volume 5689 of *LNCS*, pages 40–110.