FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

dbai

Database and Artificial Intelligence Group

Implementation of Argumentation

Stefan Woltran

G. Charwat, W. Dvořák, S. Gaggl, J. Wallner

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

ACAI'13 - July 2, 2013

Material

- Final version of slides:

  `http://www.dbai.tuwien.ac.at/research/slides/acai.pdf`

- Survey on implementation of abstract argumentation:

  `http://www.dbai.tuwien.ac.at/research/report/dbai-tr-2013-82.pdf`

Aim of this Talk

- Overview of (our) systems
- Focus on approaches based on Answer-Set Programming
- In (some) detail:
  - ASPARTIX
  - VISPARTIX
  - dynPARTIX
- After the talk: Demo-Session

Argumentation — The Big Picture

Steps

Starting point: knowledge-base

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

Argumentation — The Big Picture

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 — The Big Picture

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 — The Big Picture

Steps

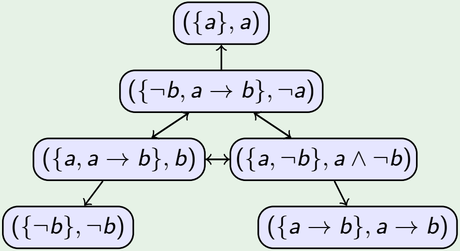
Starting point: knowledge-base

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

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



Argumentation — The Big Picture

Steps

Starting point: knowledge-base

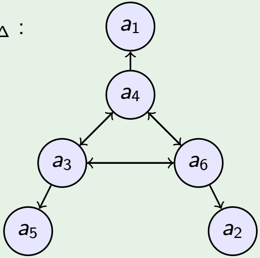
1) Form arguments
2) Identify conflicts
3) **Abstract from internal structure**
4) Resolve conflicts
5) Obtain conclusions

- Obtain an Abstract Argumentation Framework F_Δ

Example

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

F_Δ :



Argumentation — The Big Picture

Steps

Starting point: knowledge-base

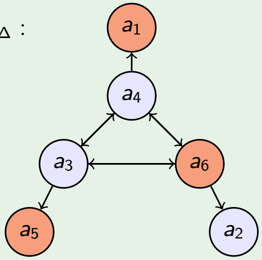
1) Form arguments
2) Identify conflicts
3) Abstract from internal structure
4) **Resolve conflicts**
5) Obtain conclusions

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

Example

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

F_Δ :



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

Argumentation — The Big Picture

Steps

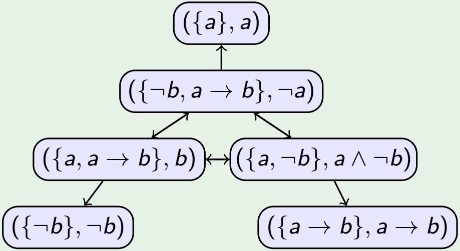
Starting point: knowledge-base

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

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



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

Important Observations

- Each of the steps computationally involved
  - design of systems from scratch vs. reduction-based method
  - in both cases: complexity adequacy important
- Two approaches for the whole process:
  - modular solutions
  - full systems

Landscape of Systems

	direct	reduction
abstract arg.	COMPARG, dynPARTIX	CONARG, **ASPARTIX**
full	TOAST, CARNEADES	**VISPARTIX**+ASPARTIX

Some other approaches follow a "mixed" approach:

- CEGARTIX
- **D-FLAT/dynPARTIX**

Answer-Set Programming

ASP Syntax

A rule r is an expression of the form

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

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;
- $B(r) = \{b_1, \ldots, b_k, \ \mathit{not}\, b_{k+1}, \ldots, \ \mathit{not}\, b_m\}$ 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.

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 π, if each $r \in \pi$ is satisfied by I.
- A non-ground rule r (resp., a program π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\pi)$).

Gelfond-Lifschitz reduct

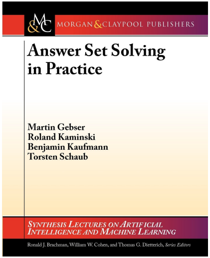
An interpretation I is an answer set of π iff it is a subset-minimal set satisfying

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

ASP - Some Remarks

- Rich rule-based language
- Well suited for combinatorial problems in NP – Guess & Check approach:
  - guess a candidate solution non-deterministically
  - check if the candidate is indeed a solution
- Even problems with high complexity (2nd level in polynomial hierarchy) can be expressed
- Advanced systems available

ASP Information + Systems



- Potassco: `http://potassco.sourceforge.net/`
- DLV: `http://www.dlvsystem.com/`

ASPARTIX Overview

Input

AF

arg(a). att(a,b).
arg(b). att(c,b).
arg(c). att(c,d).
...

Semantics

Preferred
Semi-Stable
Stage
...

ASP-Solver

gringo/clasp(D)
DLV

Answer Sets

$\{\text{in}(a), \text{in}(c), \ldots\}$,
$\{\text{in}(a), \text{in}(d), \ldots\}$

ASPARTIX Overview

Input

AF

arg(a). att(a,b).
arg(b). att(c,b).
arg(c). att(c,d).
...

Semantics

Preferred
Semi-Stable
Stage
...

ASP-Solver

gringo/clasp(D)
DLV

Answer Sets

$\{\text{in}(a), \text{in}(c), \ldots\}$,
$\{\text{in}(a), \text{in}(d), \ldots\}$

ASPARTIX Overview

Input

AF

arg(a). att(a,b).
arg(b). att(c,b).
arg(c). att(c,d).
...

Semantics

Preferred
Semi-Stable
Stage
...

ASP-Solver

gringo/clasp(D)
DLV

Answer Sets

$\{\mathrm{in}(a), \mathrm{in}(c), \ldots\}$,
$\{\mathrm{in}(a), \mathrm{in}(d), \ldots\}$

ASP Encodings

Argumentation Framework Data Base

$$\mathrm{arg(a).\ arg(b).\ arg(c).\ att(a,c).\ att(b,c).}$$

Conflict-Free Sets

$$\begin{aligned} \mathrm{in(X)} &\leftarrow \mathrm{arg(X)}, \textit{not}\ \mathrm{out(X)}. \\ \mathrm{out(X)} &\leftarrow \mathrm{arg(X)}, \textit{not}\ \mathrm{in(X)}. \\ &\leftarrow \mathrm{in(X), in(Y), att(X,Y)}. \end{aligned}$$

Stable Extensions

$$\begin{aligned} \mathrm{defeated(X)} &\leftarrow \mathrm{in(Y), att(Y,X)}. \\ &\leftarrow \mathrm{out(X)}, \textit{not}\ \mathrm{defeated(X)}. \end{aligned}$$

ASP Encodings

Argumentation Framework Data Base

$$\mathrm{arg}(\mathrm{a}).\ \mathrm{arg}(\mathrm{b}).\ \mathrm{arg}(\mathrm{c}).\ \mathrm{att}(\mathrm{a},\mathrm{c}).\ \mathrm{att}(\mathrm{b},\mathrm{c}).$$

Conflict-Free Sets

$$\begin{array}{rcl} \mathrm{in}(X) & \leftarrow & \mathrm{arg}(X), \textit{not}\ \mathrm{out}(X). \\ \mathrm{out}(X) & \leftarrow & \mathrm{arg}(X), \textit{not}\ \mathrm{in}(X). \\ & \leftarrow & \mathrm{in}(X), \mathrm{in}(Y), \mathrm{att}(X,Y). \end{array}$$

Stable Extensions

$$\begin{array}{rcl} \mathrm{defeated}(X) & \leftarrow & \mathrm{in}(Y), \mathrm{att}(Y,X). \\ & \leftarrow & \mathrm{out}(X), \textit{not}\ \mathrm{defeated}(X). \end{array}$$

ASP Encodings

Argumentation Framework Data Base

$$\mathrm{arg}(\mathrm{a}).\ \mathrm{arg}(\mathrm{b}).\ \mathrm{arg}(\mathrm{c}).\ \mathrm{att}(\mathrm{a},\mathrm{c}).\ \mathrm{att}(\mathrm{b},\mathrm{c}).$$

Conflict-Free Sets

$$\begin{array}{rcl}
\mathrm{in}(\mathrm{X}) & \leftarrow & \mathrm{arg}(\mathrm{X}), \textit{not}\ \mathrm{out}(\mathrm{X}). \\
\mathrm{out}(\mathrm{X}) & \leftarrow & \mathrm{arg}(\mathrm{X}), \textit{not}\ \mathrm{in}(\mathrm{X}). \\
 & \leftarrow & \mathrm{in}(\mathrm{X}), \mathrm{in}(\mathrm{Y}), \mathrm{att}(\mathrm{X},\mathrm{Y}).
\end{array}$$

Stable Extensions

$$\begin{array}{rcl}
\mathrm{defeated}(\mathrm{X}) & \leftarrow & \mathrm{in}(\mathrm{Y}), \mathrm{att}(\mathrm{Y},\mathrm{X}). \\
 & \leftarrow & \mathrm{out}(\mathrm{X}), \textit{not}\ \mathrm{defeated}(\mathrm{X}).
\end{array}$$

ASP Encodings

Argumentation Framework Data Base

$$\mathrm{arg}(a).\ \mathrm{arg}(b).\ \mathrm{arg}(c).\ \mathrm{att}(a,c).\ \mathrm{att}(b,c).$$

Admissible Sets

$$\begin{aligned}
\mathrm{in}(X) &\leftarrow \mathrm{arg}(X), \textit{not}\ \mathrm{out}(X).\\
\mathrm{out}(X) &\leftarrow \mathrm{arg}(X), \textit{not}\ \mathrm{in}(X).\\
&\leftarrow \mathrm{in}(X), \mathrm{in}(Y), \mathrm{att}(X,Y).\\
\mathrm{defeated}(X) &\leftarrow \mathrm{in}(Y), \mathrm{att}(Y,X).\\
\mathrm{undefended}(X) &\leftarrow \mathrm{att}(Y,X), \textit{not}\ \mathrm{defeated}(Y).\\
&\leftarrow \mathrm{in}(X), \mathrm{undefended}(X).
\end{aligned}$$

Complete Sets

$$\leftarrow \mathrm{out}(X), \textit{not}\ \mathrm{undefended}(X).$$

ASP Encodings

Argumentation Framework Data Base

$$\mathrm{arg(a).\ arg(b).\ arg(c).\ att(a,c).\ att(b,c).}$$

Admissible Sets

$$
\begin{array}{rcl}
\mathrm{in(X)} & \leftarrow & \mathrm{arg(X)}, \textit{not}\ \mathrm{out(X)}. \\
\mathrm{out(X)} & \leftarrow & \mathrm{arg(X)}, \textit{not}\ \mathrm{in(X)}. \\
 & \leftarrow & \mathrm{in(X), in(Y), att(X,Y)}. \\
\mathrm{defeated(X)} & \leftarrow & \mathrm{in(Y), att(Y,X)}. \\
\mathrm{undefended(X)} & \leftarrow & \mathrm{att(Y,X)}, \textit{not}\ \mathrm{defeated(Y)}. \\
 & \leftarrow & \mathrm{in(X), undefended(X)}.
\end{array}
$$

Complete Sets

$$\leftarrow \quad \mathrm{out(X)}, \textit{not}\ \mathrm{undefended(X)}.$$

ASP Encodings

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

Involved Encodings

Preferred Extensions (partial) module

$$
\begin{array}{rcl}
\mathrm{inN}(X) \vee \mathrm{outN}(X) & \leftarrow & \mathrm{out}(X). \\
\mathrm{inN}(X) & \leftarrow & \mathrm{in}(X). \\
\mathrm{undefeated\_upto}(X, Y) & \leftarrow & \mathrm{inf}(Y), \mathrm{outN}(X), \mathrm{outN}(Y). \\
\mathrm{undefeated\_upto}(X, Y) & \leftarrow & \mathrm{inf}(Y), \mathrm{outN}(X), \mathit{not}\ \mathrm{att}(Y, X). \\
\mathrm{undefeated\_upto}(X, Y) & \leftarrow & \mathrm{succ}(Z, Y), \mathrm{undefeated\_upto}(X, Z), \mathrm{outN}(Y). \\
\mathrm{undefeated\_upto}(X, Y) & \leftarrow & \mathrm{succ}(Z, Y), \mathrm{undefeated\_upto}(X, Z), \mathit{not}\ \mathrm{att}(Y, X). \\
\mathrm{undefeated}(X) & \leftarrow & \mathrm{sup}(Y), \mathrm{undefeated\_upto}(X, Y). \\
\mathrm{eq\_upto}(X) & \leftarrow & \mathrm{inf}(X), \mathrm{in}(X), \mathrm{inN}(X). \\
\mathrm{eq\_upto}(X) & \leftarrow & \mathrm{inf}(X), \mathrm{out}(X), \mathrm{outN}(X). \\
\mathrm{eq\_upto}(X) & \leftarrow & \mathrm{succ}(Y, X), \mathrm{in}(X), \mathrm{inN}(X), \mathrm{eq\_upto}(Y). \\
\mathrm{eq\_upto}(X) & \leftarrow & \mathrm{succ}(Y, X), \mathrm{out}(X), \mathrm{outN}(X), \mathrm{eq\_upto}(Y). \\
\mathrm{eq} & \leftarrow & \mathrm{sup}(X), \mathrm{eq\_upto}(X). \\
\mathrm{fail} & \leftarrow & \mathrm{eq}. \\
\mathrm{fail} & \leftarrow & \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{att}(X, Y). \\
\mathrm{fail} & \leftarrow & \mathrm{inN}(X), \mathrm{outN}(Y), \mathrm{att}(Y, X), \mathrm{undefeated}(Y). \\
\mathrm{inN}(X) & \leftarrow & \mathrm{fail}, \mathrm{arg}(X). \\
\mathrm{outN}(X) & \leftarrow & \mathrm{fail}, \mathrm{arg}(X). \\
 & \leftarrow & \mathit{not}\ \mathrm{fail}.
\end{array}
$$

Metasp

- Recently proposed `metasp` offers meta-programming for the gringo/claspD package
- The problem encoding is first grounded with the reify option, which outputs ground program as facts
- Next the meta encodings mirror answer-set generation, but may implement different behavior

- Meta encodings also implement subset minimization for the #minimize-statement.

Metasp Encoding

- Together with the modules for conflict-free sets and admissibility, the remaining encoding for subset maximization reduces to

Preferred Extensions

$$\pi_{adm} \cup \{\#\text{minimize}[\text{out}(X)]\}.$$

ASPARTIX

aspartix wwtf project on argumentation system page

aspartix > load >

contact help print

Specification of Input:

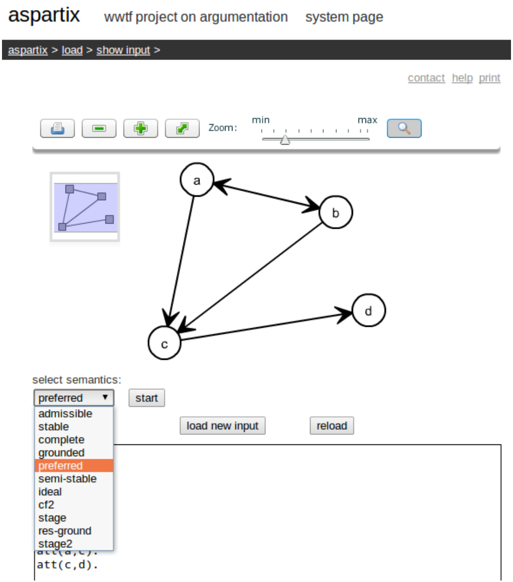
each argument "a" is defined via **arg(a).** ,
and each attack from argument "a" to argument "b" is defined via **att(a,b).**.

```
arg(a).
arg(b).
arg(c).
arg(d).
att(a,b).
att(b,a).
att(b,d).
att(c,d).
```

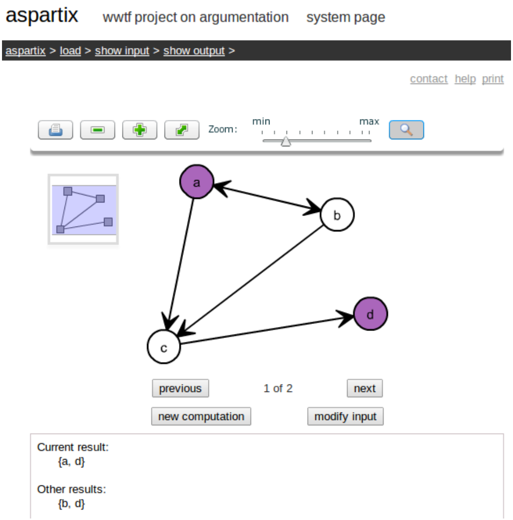
load

Problems with specifications? Look for some examples!

ASPARTIX


aspartix
wwtf project on argumentation
system page
aspartix > load > show input >
contact help print
Zoom:
min
max
a
b
c
d
select semantics:
preferred
start
admissible
stable
complete
grounded
preferred
semi-stable
ideal
cf2
stage
res-ground
stage2
load new input
reload
att(c,d).

ASPARTIX


aspartix
wwtf project on argumentation
system page
aspartix > load > show input > show output >
contact help print
Zoom:
min
max
a
b
c
d
previous
1 of 2
next
new computation
modify input
Current result:
{a, d}
Other results:
{b, d}

VISPARTIX – ASP used for generating abstract AFs

- Tool for handling instantiation process
  - forming arguments from a knowledge base
  - identifying the conflicts
- Visualization of obtained argumentation frameworks
- Based on a two-step evaluation of answer-set programs

Overall Approach

Given knowledge base $\mathcal{K}$; possible claims $\mathcal{C}$:

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

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

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

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

Example

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

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

Forming Arguments

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

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

Guess arguments:

$$\begin{aligned} \pi_{\mathrm{arg}} = \{ \quad & 1\{\ \mathrm{claim}(C) : \mathrm{cl}(C)\ \}1; \\ & 1\{\ \mathrm{fs}(FS) : \mathrm{kb}(FS)\ \}; \\ & \mathrm{formula}(C) \leftarrow \mathrm{claim}(C); \\ & \mathrm{formula}(FS) \leftarrow \mathrm{fs}(FS).\ \} \end{aligned}$$

(2) Consistency: S consistent

$$\begin{aligned} \pi_{\mathrm{consistent}} = \{ \quad & 1\{\ \mathrm{true}(\mathrm{consistent}, A), \mathrm{false}(\mathrm{consistent}, A)\ \}1 \leftarrow \mathrm{atom}(A). \\ & \leftarrow \mathrm{nomodel}(\mathrm{consistent}, FS), \mathrm{fs}(FS).\ \} \end{aligned}$$

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 saturation technique

$$
\begin{aligned}
\pi_{\text{entailment}} = \{ \quad & \text{true}(\mathit{entail}, A) \vee \text{false}(\mathit{entail}, A) \leftarrow \text{atom}(A); \\
& \text{entails\_claim} \leftarrow \text{nomodel}(\mathit{entail}, \mathit{neg}(C)), \text{claim}(C); \\
& \text{entails\_claim} \leftarrow \text{nomodel}(\mathit{entail}, \mathit{FS}), \text{fs}(\mathit{FS}); \\
& \text{true}(\mathit{entail}, A) \leftarrow \text{entails\_claim}, \text{atom}(A); \\
& \text{false}(\mathit{entail}, A) \leftarrow \text{entails\_claim}, \text{atom}(A) : \\
& \leftarrow \mathit{not}\, \text{entails\_claim}. \}
\end{aligned}
$$

- If S or C not satisfied for interpretation, we obtain entails_claim and saturate
- Otherwise, constraint $\leftarrow \mathit{not}\, \text{entails\_claim}$ 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
- 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 (π_{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

$\rightarrow$ 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:

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

Output:

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

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 $\mathrm{as}(A, fs, \cdot)$ and the predicate $\mathrm{as}(A, claim, \cdot)$

Example

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

(2) Guess exactly two arguments (π_{att})

(3) Build single support formula (π_{support})

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

Identifying Conflicts between Arguments

(4) Define Attack Types (e.g. π_{att})

- 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:

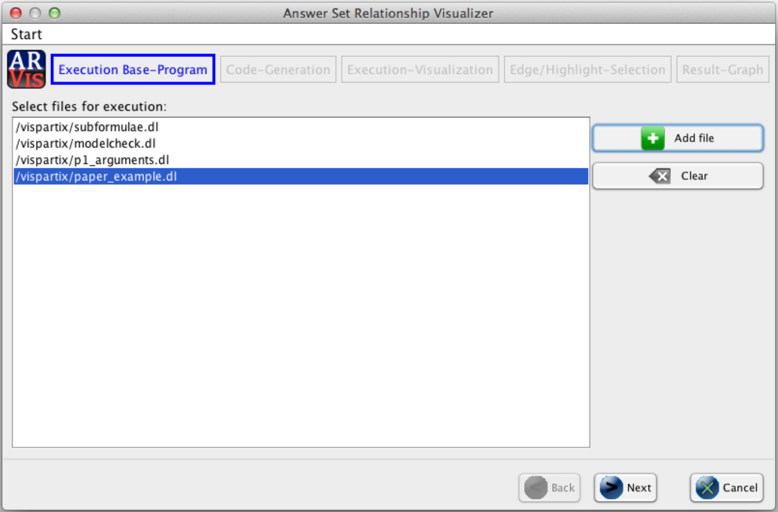
$$\pi_{\text{attacks}} = \pi_{\text{modelcheck}} \cup \pi_{\text{att}} \cup \pi_{\text{support}} \cup \pi_{\text{att}} \cup \pi_{\text{att\_sat}}$$

Visualization of Argumentation Frameworks

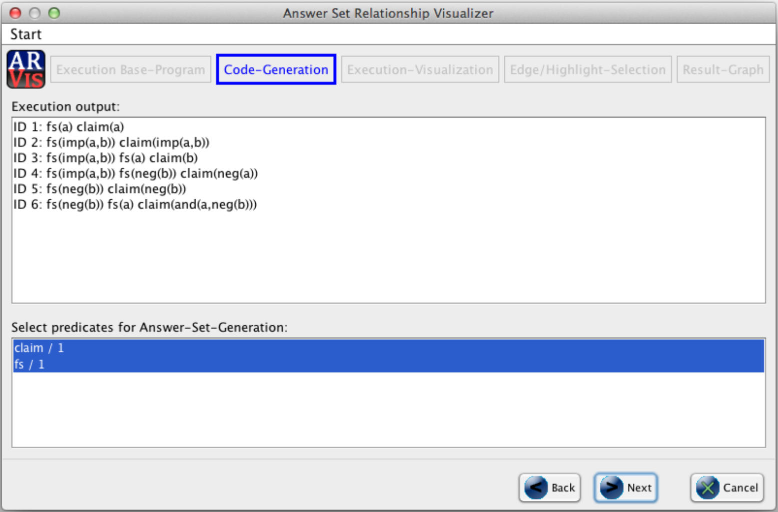
We utilize the purpose built tool ARVis (Answer set Relationship Visualizer):

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 π_{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

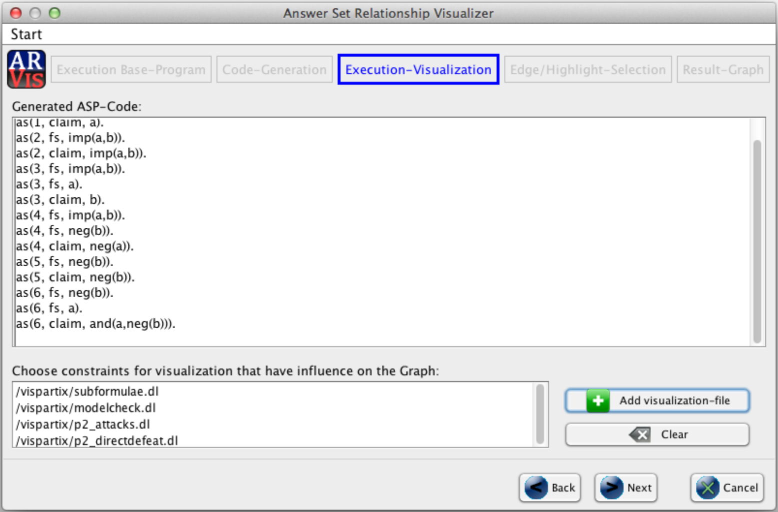
Visualization of Argumentation Frameworks



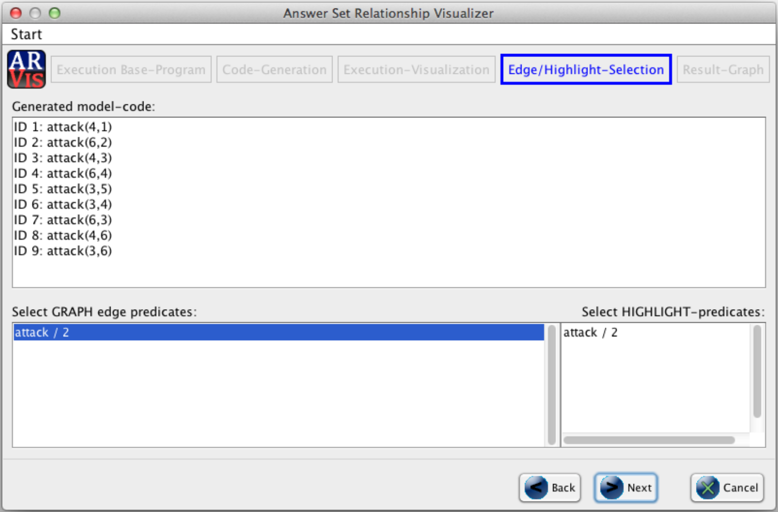
Visualization of Argumentation Frameworks



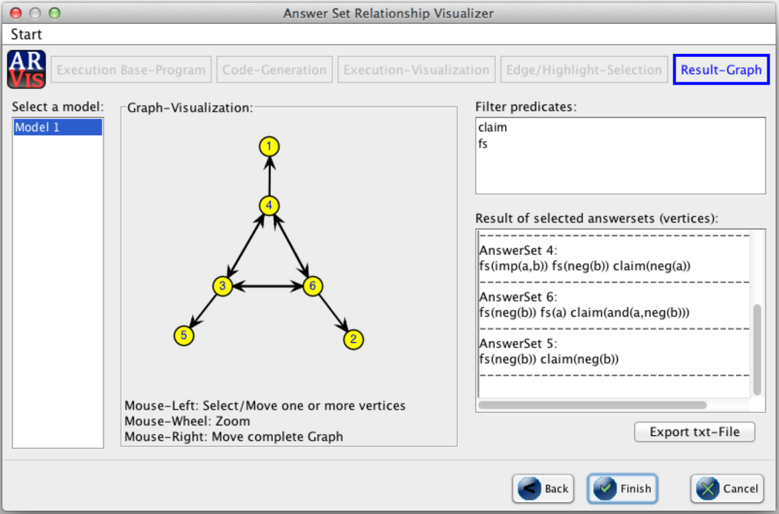
Visualization of Argumentation Frameworks



Visualization of Argumentation Frameworks



Visualization of Argumentation Frameworks



D-FLAT/dynPARTIX

Underlying Idea:

- Exploit structure of instances
- Many problems easy on tree-like graphs
- Apply *dynamic programming* on a *tree decomposition*

Systems:

- dynPARTIX: dedicated C++ system for abstract argumentation
- D-FLAT: Decompose, Guess & Check (ASP on decomposition)

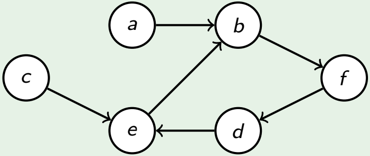
Tree Decompositions (I)

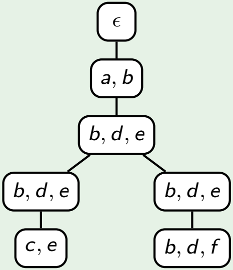
Definition

A *tree decomposition* is a tree obtained from an arbitrary graph s.t.

1. Each vertex must occur in some bag.
2. For each edge, there is a bag containing both endpoints.
3. If vertex v appears in bags of nodes n_0 and n_1, then v is also in the bag of *each node on the path* between n_0 and n_1.

Example





Tree Decompositions (II)

Definition

- *Decomposition width*: size of the largest bag (minus 1)
- Treewidth: minimum width over all possible tree decompositions

- A tree decomposition breaks an instance down into smaller parts
- Dynamic programming: Solve parts and combine partial solutions
  - Algorithms often exponential only in decomposition width
  - ... but *linear* in the input size
  - Bounded treewidth then leads to fixed-parameter tractability (FPT)

Algorithm Execution

1. Decompose instance
   - Construct a “good” tree decomposition
   - Each tree decomposition node is associated with a table
   - Each table row will correspond to partial solutions
2. Solve partial problems
   - Compute the tables in a bottom-up way
   - An ASP program is executed *for each table*
     - The ASP program is provided by the user
     - Child tables are supplied as input
     - Answer sets correspond to new table rows
3. Combine solutions

Example: Computing Stable Extensions

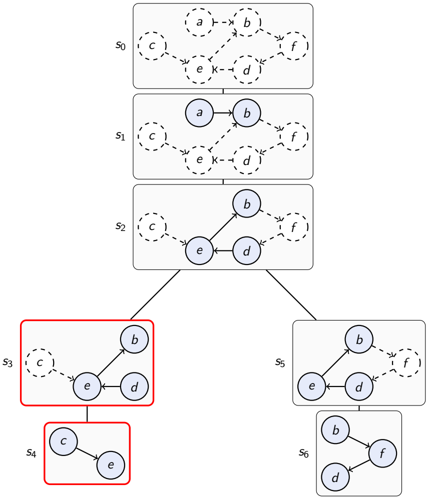


Figure: Nodes with represented AFs

s_3

b	d	e	$e_t(C)$
in	*in*	*def*	$\{\{b, c, d\}\}$
out	*in*	*def*	$\{\{c, d\}\}$
in	*out*	*def*	$\{\{b, c\}\}$
out	*out*	*def*	$\{\{c\}\}$

s_4

c	e	$e_t(C)$
in	*def*	$\{\{c\}\}$
out	*in*	$\{\{e\}\}$
out	*out*	$\{\emptyset\}$

Figure: Tables of s_3 and s_4

Implementation: D-FLAT Framework

Dynamic Programming **F**ramework with **L**ocal Execution of **A**SP on **T**ree Decompositions

1. Parses the input (graph as a set of facts) and stores the graph
2. Constructs a tree decomposition
3. In each node: Executes user-supplied algorithm with ASP solver
4. Extracts table rows from answer sets
5. Materializes the solutions

Properties

- Users only need to write an ASP program
- Communication with the user's program via special predicates

D-FLAT: Stable Extensions

User-supplied program (Stable Extensions)

```
%Exactly extend one child row per child node.
1 { extend(R) : childRow(R, N) } 1 :- childNode(N).

% For every introduced argument guess if it is in xor out/def.
{ item(map(A, in)) } :- introduced(A).

% An argument is defeated if it is not in the set and ..
% ..just introduced and attacked by an in-argument
item(map(A, def)) :- introduced(A), not item(map(A, in)), item(map(A2, in)), att(A2, A).
% ..or has already been introduced and is attacked by an in-argument.
item(map(A, def)) :- current(A), not childItem(I, map(A, in)), item(map(A2, in)), att(A2, A), extend(I).

% Copy already decided in-/def-mappings.
item(map(A, in)) :- current(A), childItem(I, map(A, in)), extend(I).
item(map(A, def)) :- current(A), childItem(I, map(A, def)), extend(I).

% Discard sets if any out-argument is removed.
:- removed(A), not childItem(I, map(A, in)), not childItem(I, map(A, def)), extend(I), childRow(I, N),
   childBag(N, A).

% Discard sets if there is a conflict.
:- item(map(A1, in)), item(map(A2, in)), att(A1, A2).

% Discard sets if the extended child rows exclude a node from the set, but the same node also is in the set.
:- extend(I), item(map(A, in)), not childItem(I, map(A, in)), current(A), childRow(I, N), childBag(N, A).
```

Summary

- Recent years have seen an emergence of argumentation systems
- Focus on systems for abstract argumentation
  - systems should cover the different semantics
  - easy-to-use interfaces important
- but what semantics / graph types are needed in the larger context?

Benchmarking – The Current State

H. Qualtinger; G. Bronner

I hob zwoar ka ohnung wo i hinfoahr
Aber dafir bin i gschwinder duat

Viennese ⇒ German:
Ich habe keine Ahnung wo ich hinfahre
Stattdessen bin ich schneller dort

German ⇒ English:
I have no idea where I am going
But for sure I am faster there

Benchmarking – The Current State

H. Qualtinger; G. Bronner

I hob zwoar ka ohnung wo i hinfoahr
Aber dafir bin i gschwinder duat

Viennese ⇒ German:
Ich habe keine Ahnung wo ich hinfahre
Stattdessen bin ich schneller dort

German ⇒ English:
I have no idea where I am going
But for sure I am faster there

Links to Systems:

- ASPARTIX http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/
- ConArg http://www.dmi.unipg.it/francesco.santini/argumentation/conarg.zip
- COMPARG http://www.ai.rug.nl/~verheij/comparg/
- Dung-O-Matic http://www.arg.dundee.ac.uk/?page_id=279
- dynPARTIX http://www.dbai.tuwien.ac.at/proj/argumentation/dynpartix
- D-FLAT http://www.dbai.tuwien.ac.at/proj/dynasp/dflat
- CEGARTIX - http://www.dbai.tuwien.ac.at/research/project/argumentation/cegartix/
- ArgKit (with Dungine) http://www.argkit.org/
- ArguLab http://heen.webfactional.com/

Links (ctd.)

- TOAST http://www.arg.dundee.ac.uk/toast/
- Vispartix http://www.dbai.tuwien.ac.at/proj/argumentation/vispartix/
- CaSAPI http://www.doc.ic.ac.uk/~ft/CaSAPI/
- Visser's Epistemic and Practical Reasoner http://www.wietskevisser.nl/research/epr/
- Carneades http://carneades.github.com/
- OVAgen http://ova.computing.dundee.ac.uk/ova-gen/
- Adam Wyner's web page http://wyner.info/LanguageLogicLawSoftware/index.php/software/

Demo Session

Thank you! And now for the demo session . . .