Formal Argumentation Approaches in TweetyProject

Matthias Thimm

University of Koblenz-Landau, Germany

SSA'20, Sep 7, 2020

TweetyProject contributors:
Lars Bengel, Linda Briesemeister, Federico Cerutti, Sebastian Franke, Nils Geilen, Anna Gessler, Timothy Gillespie, Mathias Hofer, Sebastian Homann, Tim Janus, Benedikt Knopp, Patrick Krümpelmann, Nico Potyka, Nikola Raevski, Tjitze Rienstra, Dmitry Shishkin, Stefan Tittel, Thomas Vengels, Ivan Nicola Viragine, Bastian Wolf

WeST
People and Knowledge Networks

TweetyProject: A general implementation framework

Overview:

- Tweety currently consists of 41 Java libraries dealing with different aspects of knowledge representation and artificial intelligence
- Several libraries contain basic functionalities that can be used for many different KR formalisms:
  - Abstract classes for concepts such as *Formula*, *Belief Base*, *Interpretation*, *Reasoner*,...
  - Tools for dealing with sets, graphs, mathematical expressions, mathematical optimization,...
  - Command line interface
- Basic implementations of over 25 popular KR formalisms

Design Goals:

- Ease implementation effort for KR-related approaches
- Unified development paradigm across KR formalisms
- Open source, easy access (Maven), ...

Tweety: Some statistics (as of 6 August 2020)

- 1191 SVN commits
- First SVN commit: 3 July 2010
- Current Version: 1.17
- 25 developers in total
- 4–5 active developers at any time

Code statistics (generated using David A. Wheeler's 'SLOCCount')

- Total Physical Source Lines of Code: 94,075
- Development Effort Estimate, Person-Years: 23.61
- Total Estimated Cost to Develop: 2,687,859.52 EUR

Outline

Outline

Knowledge Representation and Tweety

- Research in KR usually follows a certain template
  1. Define KR formalism (usually some logic)
     - 1.1 Syntax
     - 1.2 Semantics
  2. Define operations on KR formalism
     - 2.1 Reasoning process (calculus, tableaux, ...)
     - 2.2 Change operations (revision, update, ...)
     - 2.3 ...
  3. Analyze, evaluate and compare our approach with others
     - 3.1 Correctness, soundness
     - 3.2 Computational complexity
     - 3.3 Satisfaction of desirable properties (postulates)
     - 3.4 Expressivity
- Evaluation is usually analytical, but experimental evaluation helps for trial-and-error purposes

Overview

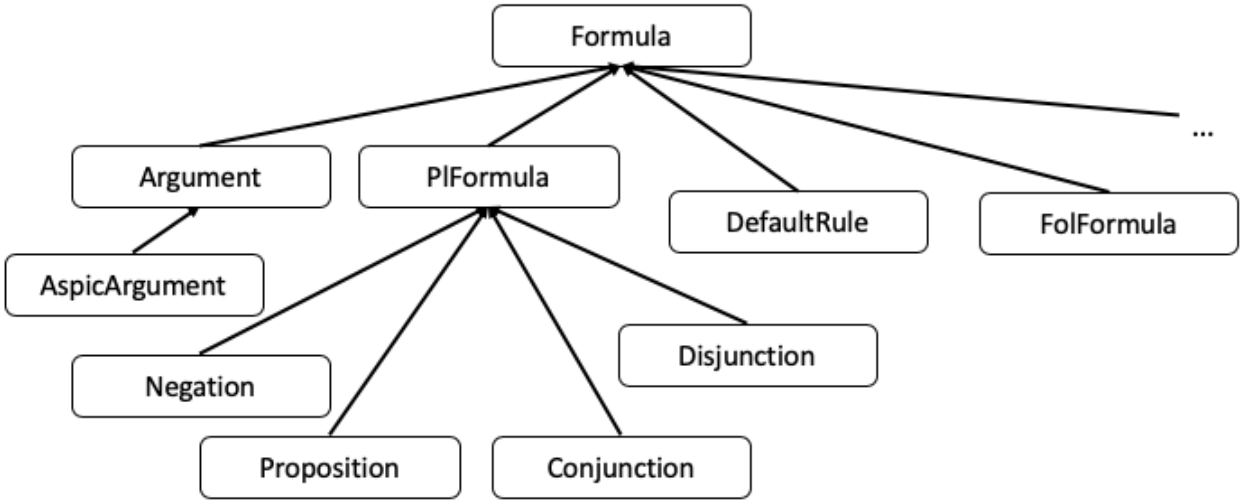
Knowledge Representation concepts are mapped 1:1 to Java classes/interfaces:

- `net.sf.tweety.commons.Formula`
- `net.sf.tweety.commons.BeliefBase`
- `net.sf.tweety.commons.Interpretation`
- `net.sf.tweety.commons.Reasoner`

Further important concepts:

- `net.sf.tweety.commons.Signature`
- `net.sf.tweety.commons.Parser` (reading files)
- `net.sf.tweety.commons.Writer` (writing files)
- Classes/interfaces for atoms, disjunctions, terms, etc.

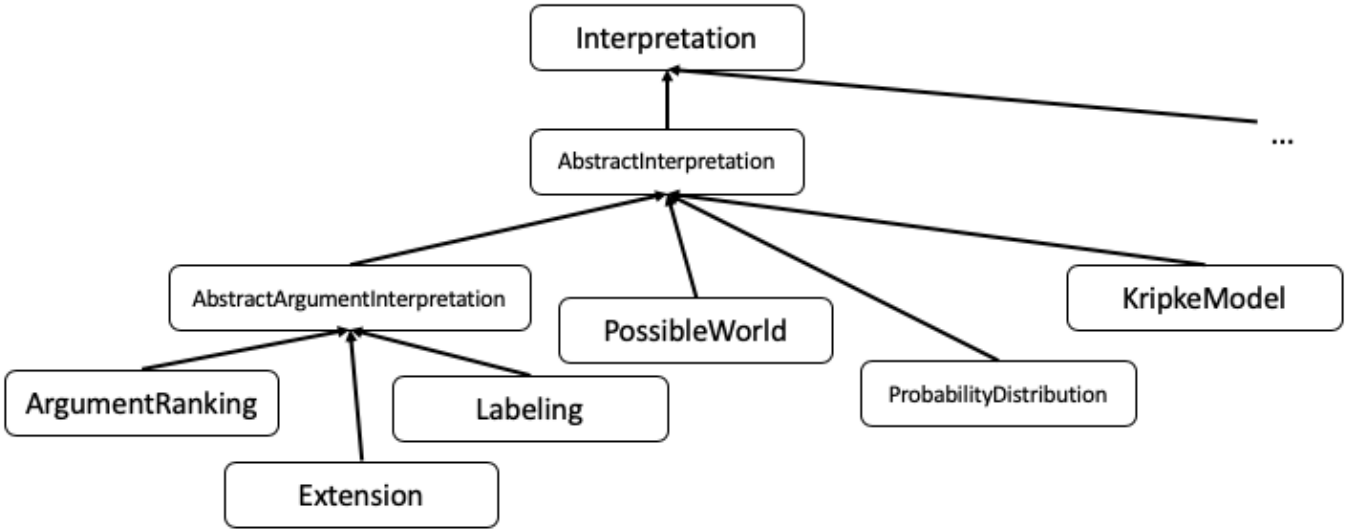
Formula




```
PlParser plParser = new PlParser();
PlFormula f = plParser.parseFormula("!a && b");
PlFormula g = plParser.parseFormula("b || c");
PlFormula h = f.combineWithAnd(g).toDnf();

FolParser folParser = new FolParser();
FolSignature sig = new FolSignature();
sig.add(new Predicate("a",2));
folParser.setSignature(sig);
FolFormula i = folParser.parseFormula
                ("forall X: (forall Y: (a(X,Y)))");
```

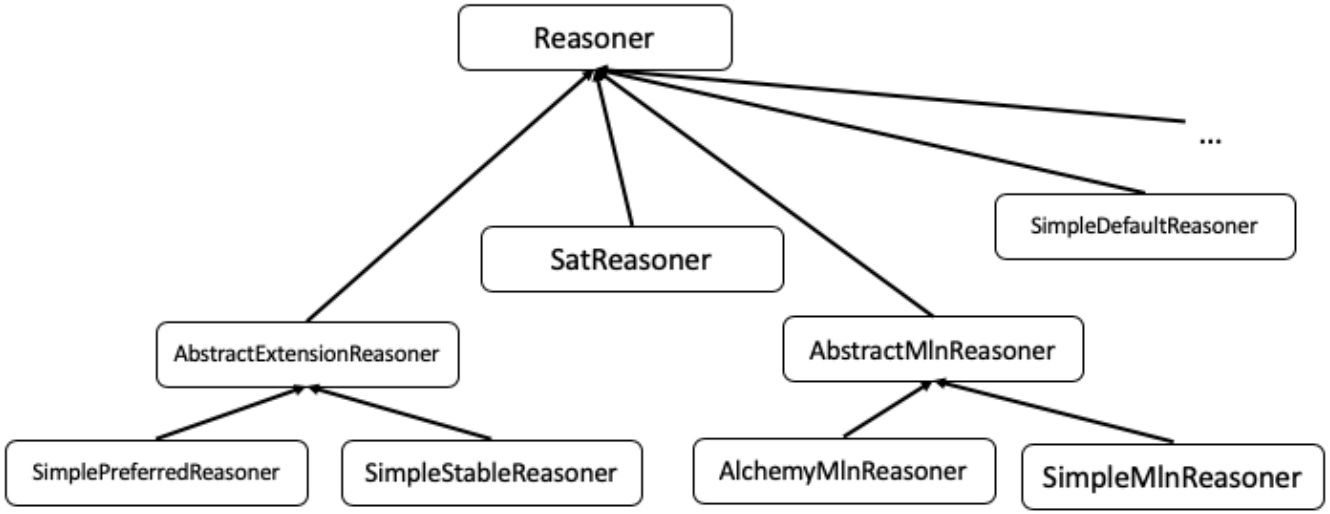
Interpretation




```
PossibleWorld w = new PossibleWorld();
w.add(new Proposition("a"));
w.add(new Proposition("b"));
System.out.println( w.satisfies(f) );

Argument a = new Argument("A");
Labeling l = new Labeling();
l.put(a,ArgumentStatus.IN);
Extension e = new Extension(l.getArgumentsOfStatus(ArgumentStatus.IN));
```

BeliefBaseReasoner




```
PlBeliefSet bs = new PlBeliefSet();
Proposition a = new Proposition("a");
Proposition b = new Proposition("b");
bs.add((PlFormula) a.complement().combineWithOr(b)); // "!a || b"
bs.add(a);
SatSolver.setDefaultSolver(new Sat4jSolver());
AbstractPlReasoner r = new SatReasoner();
System.out.println( r.query(bs, b) );
```

More Code Examples: Belief Revision

```
PlBeliefSet bs = ...;
PlFormula a = ...;

BaseRevisionOperator<PlFormula> rev =
  new LeviBaseRevisionOperator<PlFormula>(
    new KernelContractionOperator<PlFormula>(
      new RandomIncisionFunction<PlFormula>(),
      new SimplePlReasoner()),
    new DefaultBaseExpansionOperator<PlFormula>());

bs = new PlBeliefSet(rev.revise(bs, a));
```

Remark

Levi identity: $\mathcal{K} * a = (\mathcal{K} - \neg a) + a$
Kernel contraction: $\mathcal{K} - \alpha = \mathcal{K} \setminus \gamma(\mathcal{K} \perp\!\!\!\perp \alpha)$
Set of kernels (minimal proofs): $\mathcal{K} \perp\!\!\!\perp \alpha$
Incision function: γ

More Code Examples: Mathematical Optimization

```
OptimizationProblem problem =
                new OptimizationProblem(OptimizationProblem.MINIMIZE);
Variable x = new FloatVariable("x",0,1);
Variable y = new FloatVariable("y",0,1);
problem.add(new Equation(x.add(y),new FloatConstant(1)));
problem.setTargetFunction(
      new Power(x,new FloatConstant(2))
      .add(new Power(y,new FloatConstant(2))));
Solver solver = new OctaveSqpSolver();
System.out.println( solver.solve(problem) );
```

Remark

Represented optimization problem:

$\min x^2 + y^2$

s.t. $x + y = 1$

Implementation methodology 1/3

- TweetyProject strictly follows the usually recommended guidelines for object-oriented programming
  1. Represent as much as possible using interfaces and abstraction
  2. Reuse of libraries in other libraries (e. g. Probabilistic Conditional Logic extends structures from Conditional Logic which extends structures from Propositional Logic which implements abstract interfaces for basic notions)
  3. Bundle important common functionalities in abstract classes
  4. Usage of recommended design patterns (factory, observer, template method, . . .)
- TweetyProject follows the KR approach to research
  1. 1:1 mapping of logical constructs as Java classes (syntax, semantics, . . .)
  2. Logic-like composition of operators in programming code
  3. Pseudo-code in papers can be easily transformed into actually working code

Implementation methodology 2/3

- Rigorous modular integration of general methods
  - One of the first libraries was about probabilistic conditional logic; one task involves using mathematical optimization to determine a probability function
  - A general library about optimization was introduced with classes such as `OptimizationProblem`, `Equation`, ...
  - Several bridges to existing solvers were implemented (each just a couple of lines of code)
  - Library can now be used for *any* KR formalism; addition of new solvers straightforward
- TweetyProject provides lots of general applicable mathematical and logical background to easily implement even sophisticated KR formalisms

Implementation methodology 3/3

- TweetyProject provides propriety implementations of many KR *languages* (starting from propositional logic)
- TweetyProject provides *some* propriety implementations of well-known algorithms
- More importantly: existing and stable implementations of algorithms can easily be connected via *bridges*
  - Example: SAT-solvers are usually highly sophisticated
  - TweetyProject contains several bridges to existing SAT solvers; methods to convert other problems to common SAT formats (e. g. DIMACS)
  - New solvers can be added with a couple of lines of code

Outline

Outline

Installation in 30 seconds 1/4

Terminal-only, all Linux derivatives/Mac OS X (below for Ubuntu), current Java SDK should already be installed

1. Install Maven

```
$ sudo apt-get install maven
```

2. Create empty Maven project

```
$ mvn archetype:generate
    -DgroupId=mytweety.mytweetyapp
    -DartifactId=mytweetyapp
    -DarchetypeArtifactId=
        maven-archetype-quickstart
    -DinteractiveMode=false
```

3. Add TweetyProject as dependency

```
$ cd mytweetyapp
$ nano pom.xml
```

Installation in 30 seconds 2/4

3. Add TweetyProject as dependency

```
<project ...>
  ...
  <dependencies>
    ...
    <dependency>
     <groupId>net.sf.tweety</groupId>
     <artifactId>tweety-full</artifactId>
     <version>1.17</version>
    </dependency>
   ...
  </dependencies>
</project>
```

Installation in 30 seconds 3/4

4. Configure Maven for automatic dependency inclusion (recommended for beginners)

```
<project ...>
  ...
  <build>
    <plugins>
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>2.5.3</version>
            <configuration>
              <descriptorRefs>
                <descriptorRef>jar-with-dependencies</descriptorRef>
              </descriptorRefs>
            </configuration>
            <executions>
              <execution>
                <id>make-assembly</id>
                <phase>package</phase>
                <goals>
                  <goal>single</goal>
                </goals>
              </execution>
             </executions>
          </plugin>
      </plugins>
  </build>
</project>
```

Installation in 30 seconds 4/4

5. Write code

```
$ nano src/main/java/mytweety/
    mytweetyapp/App.java
```

```
package mytweety.mytweetyapp;

import net.sf.tweety.logics.pl.syntax.*;

public class App{
    public static void main( String[] args )  {
        PlFormula helloWorld =
            new Proposition("HelloWorld");
        System.out.println(helloWorld);
    }
}
```

6. Compile and run

```
$ mvn package
$ java -cp
    target/mytweetyapp-1.0-SNAPSHOT-jar-with-dependencies.jar
    mytweety.mytweetyapp.App
HelloWorld
```

Installation in Eclipse/Snapshot versions

Prerequisites for installation in Eclipse:

- Eclipse (`http://eclipse.org`)
- m2e plugin for Eclipse (`http://eclipse.org/m2e/`)

→ detailed instructions on website.

- The previous instructions always used the latest Maven version of Tweety (updated twice a year)
- In order to use the most up-to-date version we recommend using the snapshot versions from SVN with Eclipse

→ detailed instructions on website.

Outline

Libraries 1/5

General Libraries:

- Commons (`net.sf.tweety.commons`)
- Plugin (`net.sf.tweety.plugin`)
- Command Line Interface (`net.sf.tweety.cli`)
- Math (`net.sf.tweety.math`)
- Graphs (`net.sf.tweety.graphs`)

Libraries 2/5

Logic Libraries

- Logic Commons (`net.sf.tweety.logics.commons`)
- Propositional Logic (`net.sf.tweety.logics.pl`)
- First-Order Logic (`net.sf.tweety.logics.fol`)
- Conditional Logic (`net.sf.tweety.logics.cl`)
- Relational Conditional Logic (`net.sf.tweety.logics.rcl`)
- Probabilistic Conditional Logic (`net.sf.tweety.logics.pcl`)
- Quantified Boolean Formulæ (`net.sf.tweety.logics.qbf`)
- Relational Probabilistic Conditional Logic (`net.sf.tweety.logics.rpcl`)
- Reiter's Default Logic (`net.sf.tweety.logics.rdl`)
- Markov Logic (`net.sf.tweety.logics.mln`)
- Modal Logic (`net.sf.tweety.logics.ml`)
- Description Logic (`net.sf.tweety.logics.dl`)
- Business Process Modelling (`net.sf.tweety.logics.bpm`)
- Logic Translators (`net.sf.tweety.logics.translators`)

Libraries 3/5

Logic Programming Libraries

- Answer Set Programming (`net.sf.tweety.lp.asp`)
- Dynamics in Answer Set Programming (`net.sf.tweety.lp.asp.beliefdynamics`)
- Nested Logic Programming (`net.sf.tweety.lp.nlp`)

Libraries 4/5

Argumentation Libraries:

- Abstract Argumentation (`net.sf.tweety.arg.dung`)
- Assumption-based Argumentation (`net.sf.tweety.arg.aba`)
- Abstract Dialectical Frameworks (`net.sf.tweety.arg.adf`)
- ASPIC$^+$ (`net.sf.tweety.arg.aspic`)
- Bipolar Argumentation (`net.sf.tweety.arg.bipolar`)
- Deductive Argumentation (`net.sf.tweety.arg.deductive`)
- Social Abstract Argumentation (`net.sf.tweety.arg.social`)
- Structured Argumentation Frameworks (`net.sf.tweety.arg.saf`)
- Defeasible Logic Programming (`net.sf.tweety.arg.delp`)
- Logic Programming Argumentation (`net.sf.tweety.arg.lp`)
- Probabilistic Argumentation (`net.sf.tweety.arg.prob`)
- Rankings for Argumentation (`net.sf.tweety.arg.rankings`)

Libraries 5/5

Agent Libraries:

- Agents (`net.sf.tweety.agents`)
- Dialogues (`net.sf.tweety.agents.dialogues`)

Other Libraries:

- Action and Change (`net.sf.tweety.action`)
- Belief Dynamics (`net.sf.tweety.beliefdynamics`)
- Machine Learning (`net.sf.tweety.machinelearning`)
- Preferences (`net.sf.tweety.preferences`)
- Web (`net.sf.tweety.web`)

Outline

Packages

- Abstract Argumentation (`net.sf.tweety.arg.dung`)
- Assumption-based Argumentation (`net.sf.tweety.arg.aba`)
- Abstract Dialectical Frameworks (`net.sf.tweety.arg.adf`)
- ASPIC$^+$ (`net.sf.tweety.arg.aspic`)
- Bipolar Argumentation (`net.sf.tweety.arg.bipolar`)
- Deductive Argumentation (`net.sf.tweety.arg.deductive`)
- Social Abstract Argumentation (`net.sf.tweety.arg.social`)
- Structured Argumentation Frameworks (`net.sf.tweety.arg.saf`)
- Defeasible Logic Programming (`net.sf.tweety.arg.delp`)
- Logic Programming Argumentation (`net.sf.tweety.arg.lp`)
- Probabilistic Argumentation (`net.sf.tweety.arg.prob`)
- Rankings for Argumentation (`net.sf.tweety.arg.rankings`)

Also relevant:

- Agent Dialogues (`net.sf.tweety.agents.dialogues`)

Packages

- **Abstract Argumentation** (`net.sf.tweety.arg.dung`)
- Assumption-based Argumentation (`net.sf.tweety.arg.aba`)
- Abstract Dialectical Frameworks (`net.sf.tweety.arg.adf`)
- **ASPIC$^+$** (`net.sf.tweety.arg.aspic`)
- Bipolar Argumentation (`net.sf.tweety.arg.bipolar`)
- **Deductive Argumentation** (`net.sf.tweety.arg.deductive`)
- Social Abstract Argumentation (`net.sf.tweety.arg.social`)
- Structured Argumentation Frameworks (`net.sf.tweety.arg.saf`)
- Defeasible Logic Programming (`net.sf.tweety.arg.delp`)
- Logic Programming Argumentation (`net.sf.tweety.arg.lp`)
- Probabilistic Argumentation (`net.sf.tweety.arg.prob`)
- **Rankings for Argumentation** (`net.sf.tweety.arg.rankings`)

Also relevant:

- Agent Dialogues (`net.sf.tweety.agents.dialogues`)

Abstract Argumentation

Definition (Abstract Argumentation Framework)

An *abstract argumentation framework* AF is a tuple $\text{AF} = (\text{Arg}, \rightarrow)$ with arguments Arg and an attack relation $\rightarrow \subseteq \text{Arg} \times \text{Arg}$ [Dung,1995].

An extension E is a set $E \subseteq \text{Arg}$ and is supposed to model a "plausible and jointly acceptable" set of arguments.

Definition

E is *admissible* iff

1. for all $\mathcal{A}, \mathcal{B} \in E$ it is not the case that $\mathcal{A} \rightarrow \mathcal{B}$,
2. for all $\mathcal{A} \in E$, if $\mathcal{B} \rightarrow \mathcal{A}$ then there is $\mathcal{C} \in E$ with $\mathcal{C} \rightarrow \mathcal{B}$

and it is *complete* if additionally

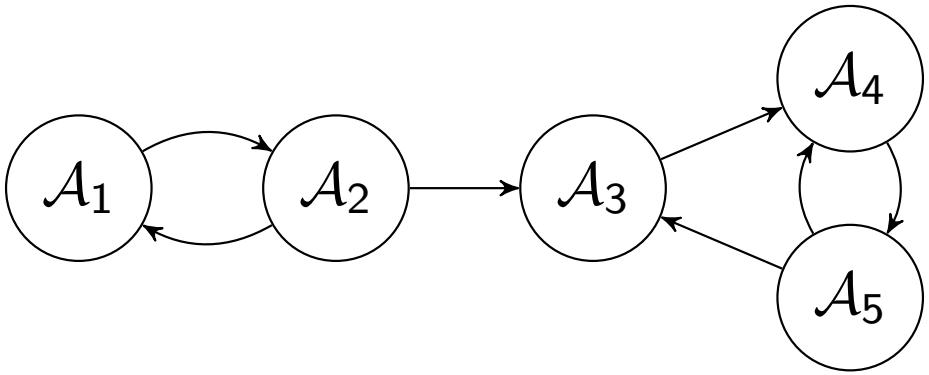
3. every argument $\mathcal{C}$ that is *defended* by E, belongs to E

Semantics

Definition

- E is *grounded* if and only if E is minimal (wrt. set inclusion).
- E is *preferred* if and only if E is maximal (wrt. set inclusion).
- E is *stable* if and only if E attacks all arguments $\text{Arg} \setminus E$.
- ...

Example



$E = \{\mathcal{A}_1, \mathcal{A}_5\}$ is admissible, complete, preferred, and stable.

$E' = \emptyset$ is admissible, complete, and grounded.

Abstract Argumentation in TweetyProject 1/2

Creating and manipulating argumentation frameworks:

```
DungTheory at = new DungTheory();
Argument a = new Argument("a");
Argument b = new Argument("b");
at.add(a,b);
Attack att1 = new Attack(a,b);
at.add(att1);
```

Parsing files in APX format:

```
DungTheory at = new ApxParser()
    .parseBeliefBaseFromFile(file);
```

Abstract Argumentation in TweetyProject 2/2

Computing extensions:

```
DungTheory at = ...

SimpleCompleteReasoner r = new SimpleCompleteReasoner();
System.out.println(r.getModels(at));
```

- Supported semantics: complete, grounded, preferred, stable, semi-stable, CF2, stage, stage2, ideal, SCF2, SCOOC-naive, semi-qualified, weak admissibility
- Standard reasoner returns objects of type `Extension` (set of arguments)
- Conversion to `Labeling` also possible
- Different strategies for reasoners available: based on semantical definitions, SAT reductions, exploitation of SCC structure
- Also available: general bridge to include any probo-compatible solver (the interface standard of the argumentation competition)

Demo session

Outline

ASPIC+ 1/3

Disclaimer: I will only talk about a much simplified propositional fragment of ASPIC+

Definition

An ASPIC+ knowledge base $\mathcal{K}$ is a pair $\mathcal{K} = (\mathcal{K}_s, \mathcal{K}_d)$ with

- $\mathcal{K}_s$ is a set of strict rules of the form $\phi_1, \ldots, \phi_n \rightarrow \psi$
- $\mathcal{K}_d$ is a set of defeasible rules of the form $\phi_1, \ldots, \phi_n \Rightarrow \psi$

Definition

An argument $\mathcal{A}$ in $\mathcal{K}$ is a tuple $\mathcal{A} = (A, \psi)$ s. t.

- $A \subseteq \mathcal{K}_s \cup \mathcal{K}_d$
- A "entails" ϕ "properly"

ASPIC+ 2/3

Definition

An argument $\mathcal{A} = (A, \psi)$ attacks an argument $\mathcal{B} = (B, \psi')$ if $\neg\psi$ is the conclusion of some rule in B (this is just one version of attack!).

Example

Let $\mathcal{K}$ be given via

$$\Rightarrow a \qquad \Rightarrow b \qquad \Rightarrow d$$
$$a \Rightarrow c \qquad b \Rightarrow \neg c \qquad c, d \Rightarrow e$$

Then

$$\mathcal{A} = (\{\Rightarrow a; \Rightarrow d; a \Rightarrow c; c, d \Rightarrow e\}, e)$$
$$\mathcal{B} = (\{\Rightarrow b; b \Rightarrow \neg c\}, \neg c)$$

are arguments and $\mathcal{B}$ attacks $\mathcal{A}$.

ASPIC+ 3/3

Reasoning in ASPIC+ is then defined through the following procedure:

1. Determine all arguments **A** in $\mathcal{K}$
2. For each pair $A, B \in \mathbf{A}$ check whether A attacks B
3. Construct an abstract argumentation framework AF from **A**
4. Determine the set of acceptable arguments in AF (e. g. all skeptically accepted arguments wrt. preferred semantics)
5. Accept all conclusions of these acceptable arguments in $\mathcal{K}$.

Demo session

Outline

Deductive Argumentation

Approach for *structured argumentation*: Deductive Argumentation [Besnard, Hunter; 2001].

Let Φ be a set of (propositional) sentences.

Definition

An *argument* $\mathcal{A}$ for a sentence α is a tuple $\mathcal{A} = \langle \Psi, \alpha \rangle$ with $\Psi \subseteq \Phi$ that satisfies

1. $\Psi \not\vdash \bot$
2. $\Psi \vdash \alpha$, and
3. there is no $\Psi' \subsetneq \Psi$ with $\Psi' \vdash \alpha$

For an argument $\mathcal{A} = \langle \Psi, \alpha \rangle$ we say that α is the *claim* of $\mathcal{A}$ and Ψ is the *support* of $\mathcal{A}$.

Undercuts and argument trees

Definition

An argument $\mathcal{A} = \langle \Psi, \alpha \rangle$ is an *undercut* for an argument $\mathcal{B} = \langle \Phi, \beta \rangle$ if and only if $\alpha = \neg(\phi_1 \wedge \ldots \wedge \phi_n)$ for some $\phi_1, \ldots, \phi_n \subseteq \Phi$.

Definition

An *argument tree* $\tau_\Phi(\alpha)$ for α in Φ is a tree where the nodes are arguments and that satisfies

1. the root is an argument for α in Φ,
2. for every path $[\langle \Phi_1, \alpha_1 \rangle, \ldots, \langle \Phi_n, \alpha_n \rangle]$ in $\tau_\Phi(\alpha)$ it holds that $\Phi_n \not\subseteq \Phi_1 \cup \ldots \cup \Phi_{n-1}$, and
3. the children $\mathcal{B}_1, \ldots, \mathcal{B}_m$ of a node $\mathcal{A}$ consist of all undercuts for $\mathcal{A}$ that obey 2.).

Let $\mathcal{T}(\mathrm{At})$ be the set of all argument trees.

Remark: "undercut" in item 3.) should be "canonical undercut"

Argument structures

Definition

The *argument structure* $\Gamma_\Phi(\alpha)$ for α with respect to Φ is the tuple $\Gamma_\Phi(\alpha) = (\mathcal{P}, \mathcal{C})$ such that $\mathcal{P}$ is the set of argument trees for α in Φ and $\mathcal{C}$ is the set of arguments trees for $\neg\alpha$ in Φ.

Definition

A *categorizer* γ is a function $\gamma : \mathcal{T}(\mathsf{At}) \rightarrow \mathbb{R}$.

Definition

An *accumulator* κ is a function $\kappa : \mathfrak{PP}(\mathbb{R}) \times \mathfrak{PP}(\mathbb{R}) \rightarrow \mathbb{R}$ ($\mathfrak{PP}(S)$ is the set of multi-sets of S).

Acceptance

Φ *accepts* a sentence α with respect to a categorizer γ and an accumulator κ, denoted by $\Phi \mid\sim_{\kappa,\gamma} \alpha$ if and and only if

$$\kappa(\gamma(\Gamma_\Phi(\alpha))) > 0$$

Demo session

Outline

Ranking-based Semantics 1/2

Definition

A *ranking-based semantics* R assigns to each argumentation framework $\text{AF} = (\text{Arg}, \rightarrow)$ a relation (usually a total preorder) $\preceq^R_{\text{AF}} \subseteq \text{Arg} \times \text{Arg}$.

Idea: $\mathcal{A} \preceq^R_{\text{AF}} \mathcal{B}$ iff $\mathcal{A}$ is at least as acceptable as $\mathcal{B}$.

Related notion:

Definition

A *graded semantics* G assigns to each argumentation framework $\text{AF} = (\text{Arg}, \rightarrow)$ a function $G_{\text{AF}} : \text{Arg} \rightarrow \mathbb{R}$.

Idea: large values $G_{\text{AF}}(\mathcal{A})$ indicate high acceptability of $\mathcal{A}$.

Remark

Every graded semantics G induces a ranking-based semantics R_G via

$$\mathcal{A} \preceq^R_{\text{AF}} \mathcal{B} \qquad \textit{iff} \qquad G(\mathcal{A}) \geq G(\mathcal{B})$$

Ranking-based Semantics 2/2

Approaches:

- Categoriser semantics [Besnard, Hunter; 2001], [Pu et al.; 2014]
- Tuples-based [Cayrol, Lagasquie-Schiex; 2005]
- Strategy-based semantics [Matt, Toni; 2008]
- Fuzzy-based semantics [Da Costa Pereira et al.; 2011]
- Discussion-based semantics [Amgoud, Ben-Naim; 2013]
- Burden-based semantics [Amgoud, Ben-Naim; 2013]
- Iterated graded defense semantics [Grossi, Modgil; 2015]
- Probabilistic Graded Semantics [Thimm et al.; 2018]
- ...

Ranking-based Semantics 2/2

Approaches:

- Categoriser semantics [Besnard, Hunter; 2001], [Pu et al.; 2014]
- Tuples-based [Cayrol, Lagasquie-Schiex; 2005]
- Strategy-based semantics [Matt, Toni; 2008]
- Fuzzy-based semantics [Da Costa Pereira et al.; 2011]
- Discussion-based semantics [Amgoud, Ben-Naim; 2013]
- Burden-based semantics [Amgoud, Ben-Naim; 2013]
- Iterated graded defense semantics [Grossi, Modgil; 2015]
- **Probabilistic Graded Semantics [Thimm et al.; 2018]**
- ...

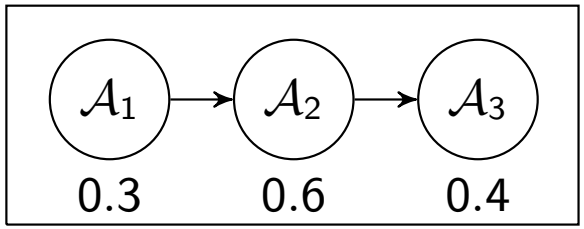
Probabilistic Argumentation Frameworks 1/2

Definition

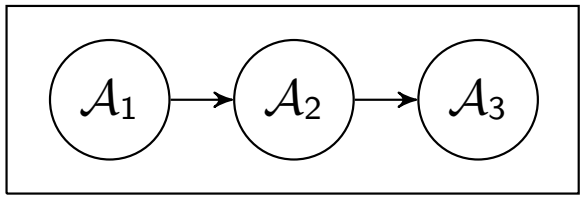
A *probabilistic argumentation framework* PAF is a triple $\text{PAF} = (\text{Arg}, \rightarrow, P)$ where $(\text{Arg}, \rightarrow)$ is an abstract argumentation framework and P is a function $P : \text{Arg} \rightarrow [0, 1]$.

See [Li et al.; 2011], [Hunter; 2014], [Hunter; 2016].

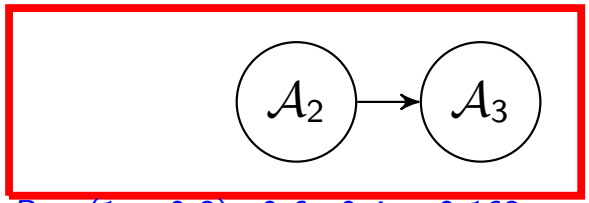
Probabilistic Argumentation Frameworks 2/2



$$P^{\text{PAF}}(\mathcal{A}_2) = 0.168 + 0.252 = 0.42$$

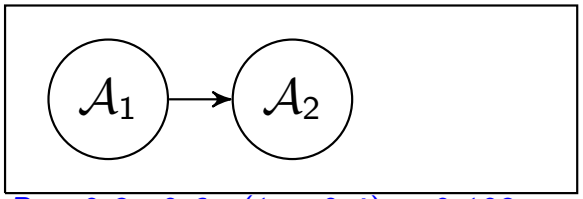


$P = 0.3 \cdot 0.6 \cdot 0.4 = 0.072$

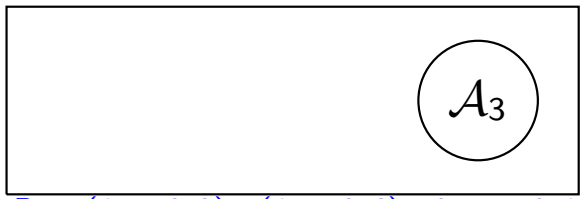


$P = (1 - 0.3) \cdot 0.6 \cdot 0.4 = 0.168$

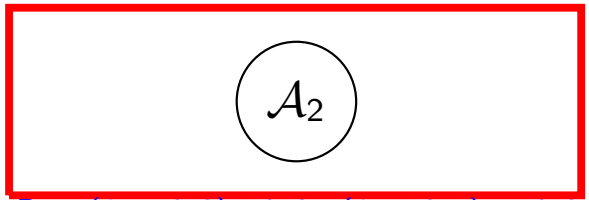
$P = 0.3 \cdot (1 - 0.6) \cdot 0.4 = 0.048$



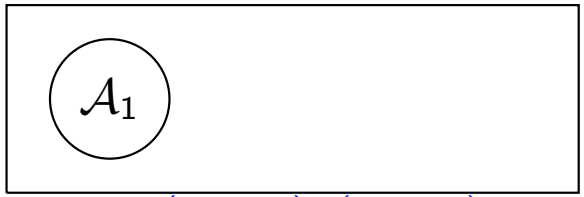
$P = 0.3 \cdot 0.6 \cdot (1 - 0.4) = 0.108$



$P = (1 - 0.3) \cdot (1 - 0.6) \cdot 0.4 = 0.112$



$P = (1 - 0.3) \cdot 0.6 \cdot (1 - 0.4) = 0.252$



$P = 0.3 \cdot (1 - 0.6) \cdot (1 - 0.4) = 0.072$

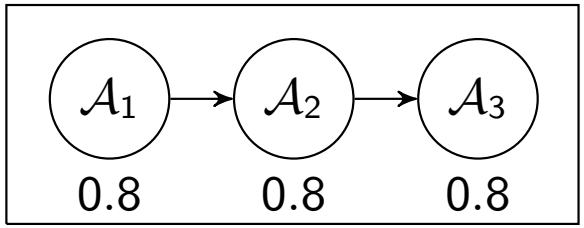
$P = (1 - 0.3) \cdot (1 - 0.6) \cdot (1 - 0.4) = 0.168$

Probabilistic Graded Semantics

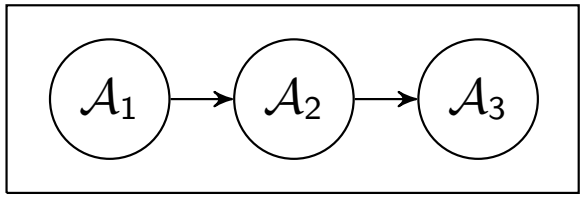
Idea:

- Let $p \in [0, 1]$
- AF[p]: PAF $(\text{Arg}, \rightarrow, P)$ with $P(\mathcal{A}) = p$ for all $\mathcal{A} \in \text{Arg}$
- Use probabilities $P^{\text{AF}[p]}(\cdot)$ as acceptability degrees

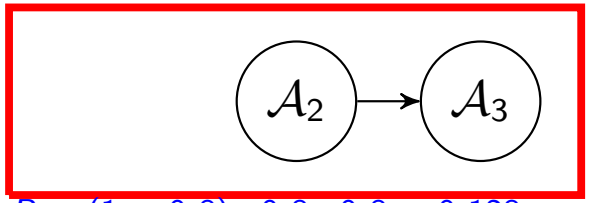
Probabilistic Graded Semantics - Example



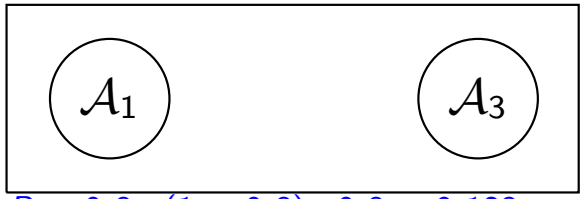
$$P^{\text{PAF}}(\mathcal{A}_1) = 0.8$$
$$P^{\text{PAF}}(\mathcal{A}_2) = 0.128 + 0.032 = 0.16$$
$$P^{\text{PAF}}(\mathcal{A}_3) = 0.672$$



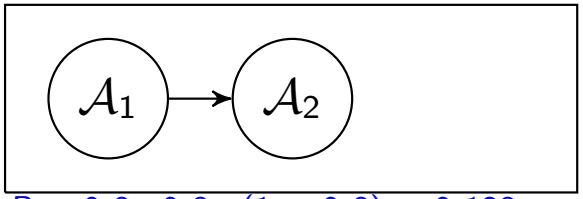
$P = 0.8 \cdot 0.8 \cdot 0.8 = 0.512$



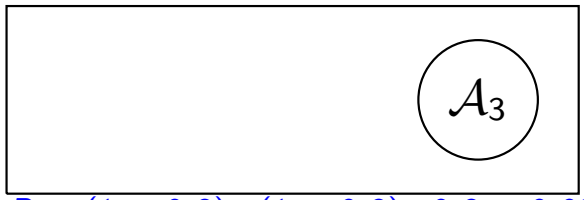
$P = (1 - 0.8) \cdot 0.8 \cdot 0.8 = 0.128$



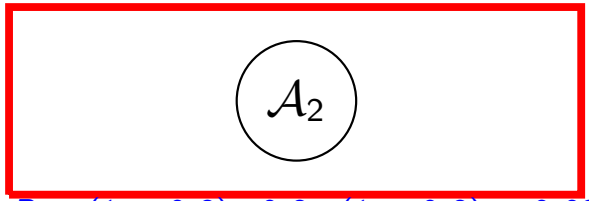
$P = 0.8 \cdot (1 - 0.8) \cdot 0.8 = 0.128$



$P = 0.8 \cdot 0.8 \cdot (1 - 0.8) = 0.128$

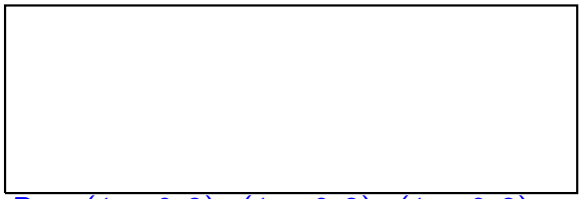


$P = (1 - 0.8) \cdot (1 - 0.8) \cdot 0.8 = 0.032$



$P = (1 - 0.8) \cdot 0.8 \cdot (1 - 0.8) = 0.032$

$P = 0.8 \cdot (1 - 0.8) \cdot (1 - 0.8) = 0.032$



$P = (1 - 0.8) \cdot (1 - 0.8) \cdot (1 - 0.8) = 0.008$

Rationality postulates 1/2

Ranking-based/graded semantics are usually evaluated with a series of rationality postulates, cf. [Amgoud, Ben-Naim; 2013], [Bonzon et al.; 2016].

- *Abstraction*
  The ranking on arguments should be defined only on the basis of the attacks between them.
- *Independence*
  The ranking between two arguments $\mathcal{A}$ and $\mathcal{B}$ should be independent of any argument that is neither connected to $\mathcal{A}$ nor to $\mathcal{B}$.
- *Addition of an Attack Branch*
  Adding a new attack line to any argument degrades its ranking.
- *Self-contradiction*
  A self-attacking argument should be ranked lower than any non self-attacking argument.
- ...

Rationality postulates 2/2

Properties	Cat	Dbs	Bbs	α-Bbs	FL	CS	$Propa_{\epsilon}$	$Propa_{1+\epsilon}$	$Propa_{1\to\epsilon}$	Tuples	M&T	IGD	Gr
Abs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
In	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓
VP	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	×
DP	✓	✓	✓	✓	×	✓	✓	✓	✓	×	×	×	×
CT	✓	✓	✓	✓	✓	✓	$\checkmark_M$	$\checkmark_M$	×	×	×	×	×
SCT	✓	✓	✓	✓	×	✓	$\checkmark_M$	$\checkmark_M$	×	×	×	×	×
CP	×	✓	✓	×	×	×	×	×	×	×	×	×	×
QP	×	×	×	×	✓	×	×	×	×	×	×	×	×
DDP	×	×	✓	×	×	×	×	✓	✓	×	×	×	×
SC	×	×	×	×	×	×	×	×	×	×	✓	×	×
$\oplus$DB	×	×	×	×	×	×	×	×	×	×	×	×	×
+DB	×	×	×	×	×	×	×	×	✓	✓	×	×	×
$\uparrow$AB	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×	×	×
$\uparrow$DB	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×	×	×
+AB	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×	✓	×
Tot	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	×	✓
NaE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AE	✓	✓	✓	✓	?	✓	✓	✓	✓	✓	?	?	✓
OE	✓	✓	✓	✓	✓	✓	$\checkmark_M$	$\checkmark_M$	$\checkmark_M$	✓	×	?	×
AvsFD	×	×	×	×	✓	×	×	✓	✓	✓	✓	×	✓

[Jérôme Delobelle. Ranking-based Semantics for Abstract Argumentation. 2017]

Demo session

Outline

Contributing to TweetyProject

- TweetyProject is a collaborative research project
- Contribute with
  - bugfixes to existing libraries
  - new implementations/alternatives to extend existing libraries
  - completely new libraries
- Just register at SourceForge and provide your username to Matthias Thimm (`thimm@uni-koblenz.de`)
- Write-access to the repository will be enabled afterwards

Final Remarks and further work

TweetyProject is . . .

- . . . a multi-purpose framework for Knowledge Representation
- . . . open source and licensed under GNU LGPL v3.0
- . . . being constantly improved

Current Work:

- Abstract Dialectical Frameworks library
- Further ranking-based semantics
- Learning argumentation frameworks

Future Work:

- Web interface
- TweetyScript
- More formalisms

Thank you for your attention

Join and participate: `http://tweetyproject.org`