

Implementing KR Approaches with Tweety

Matthias Thimm

Institute for Web Science and Technologies, University of Koblenz-Landau

July 25, 2015

How to do research in (theoretical) KR:

1. Have a great idea
2. Formalize idea
3. Proof properties
4. Write paper



How to do research in practical CS:

1. Have a great idea
2. Implement idea
3. Do empirical evaluation
4. Write paper



- ▶ Even theoretical work can benefit from implementations
 - ▶ allows other people to easily do examples
 - ▶ proof-of-concept to show feasibility
 - ▶ compare behavior with other approaches, not just with toy examples
- ▶ KR2012: 20 of 71 (long and short) papers mentioned implementations ($\approx 28\%$, compare to e. g. Semantic Web conferences with $> 90\%$)



- ▶ What is the problem?
 - ▶ approaches to complex to compute
 - ▶ Not with todays computational resources (proof-of-concept!)
 - ▶ “research is not about implementation”
 - ▶ KR is meant to be about applications (mainly)
 - ▶ lack of implementation skills
 - ▶ No excuse
 - ▶ no time for implementation
 - ▶ No excuse
- ▶ Remark: DL research is a bit of an exception



Tweety: A general implementation framework

Overview:

- ▶ Tweety currently consists of 32 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 15 popular KR formalisms

Design Goals:

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



Outline

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- 1 Introduction
 - Purpose and Overview of Tweety
 - Related Works
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ Tweety is a collection of Java libraries to aid researchers in *knowledge representation* (mainly) in implementation work
- ▶ Definition “Knowledge Representation”:
Knowledge representation and reasoning (KR) is the field of artificial intelligence (AI) dedicated to representing information about the world in a form that a computer system can utilize to solve complex tasks such as diagnosing a medical condition. [Wikipedia]

- ▶ 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 analytically, but experimental evaluation helps for trial-and-error purposes

- ▶ KR research is well-structured
- ▶ KR formalisms share the same structure
- ▶ This makes it easy for object-oriented implementation of *lots* of formalisms
 - ▶ Interfaces for common concepts (e. g. Formula or Reasoner)
 - ▶ Specific implementations for each logic
 - ▶ Modularity, refinement, composition of different aspects
- ▶ The use of Java and the object-oriented programming paradigm (other than functional or logical paradigms) has several advantages
 1. Everyone knows Java (or: more people know Java than Haskell)
 2. Undergraduate students know Java as well
 3. Everyone can run Java programs (easy setup, ...)
 4. *insert your favorite Java argument*

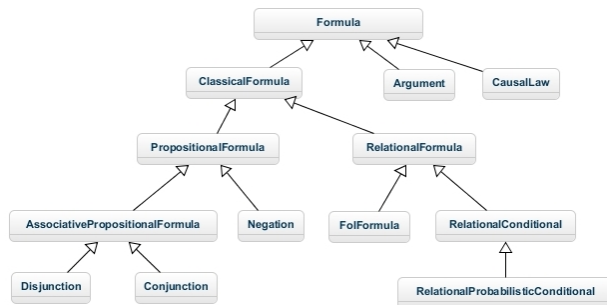
- 1 Introduction
 - Purpose and Overview of Tweety
 - Related Works
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

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.EntailmentRelation`
- ▶ `net.sf.tweety.commons.Reasoner`

Further important concepts:

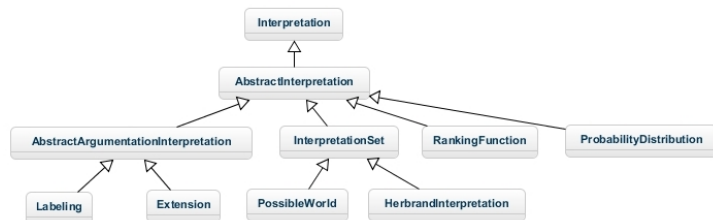
- ▶ `net.sf.tweety.commons.Signature`
- ▶ `net.sf.tweety.commons.Parser` (reading files)
- ▶ `net.sf.tweety.commons.Writer` (writing files)
- ▶ Classes/interfaces for atoms, disjunctions, terms, etc.



Example

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

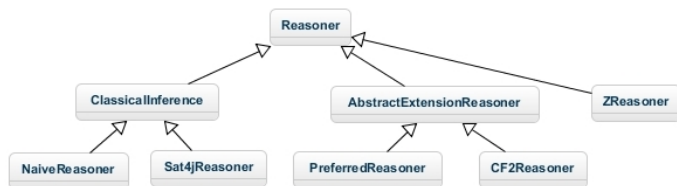
```
FolFormula i = new FolParser().parseFormula  
    ("forall X: (forall Y: A(X,Y))");
```



Example

```
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.getArgumentOfStatus(ArgumentStatus.IN));
```



Example

```
PlBeliefSet bs = new PlBeliefSet();  
Proposition a = new Proposition("a");  
Proposition b = new Proposition("b");  
bs.add(a.complement().combineWithOr(b)); // "!a || b"  
bs.add(a);  
Reasoner r = new Sat4jReasoner(bs);  
System.out.println( r.query(b) );
```

More Code Examples: Belief Revision

```
PlBeliefSet bs = ...;
PropositionalFormula a = ...;

BaseRevisionOperator<PropositionalFormula> rev =
    new LeviBaseRevisionOperator<PropositionalFormula>(
        new KernelContractionOperator<PropositionalFormula>(
            new RandomIncisionFunction<PropositionalFormula>(),
            new ClassicalEntailment()),
        new DefaultBaseExpansionOperator<PropositionalFormula>());

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


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

Remark

Represented optimization problem:

$$\min x^2 + y^2$$

$$\text{s.t. } x + y = 1$$

- 1 Introduction
 - Purpose and Overview of Tweety
 - Related Works
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

General purpose

- ▶ Tweety is a general-purpose collection of libraries addressing various aspects of KR and AI in general
- ▶ To the best of my knowledge there is no other project with the same broad scope

Other works

- ▶ BContractor
 - ▶ General framework for belief revision/contraction
 - ▶ <https://bitbucket.org/renatolundberg/bcontractor>
- ▶ PRISM
 - ▶ Framework for modelling probabilistic systems
 - ▶ <http://www.prismmodelchecker.org>
- ▶ Alchemy
 - ▶ Reasoning engine for Markov Logic Networks
 - ▶ <http://alchemy.cs.washington.edu>

Other works

- ▶ SATEN
 - ▶ General framework for belief revision/contraction
 - ▶ *no valid URL found*
- ▶ KReator
 - ▶ IDE for probabilistic relational approaches (SRL)
 - ▶ <http://kreator-ide.sourceforge.net>
- ▶ COBA 2.0
 - ▶ Belief change system
 - ▶ <http://www.cs.sfu.ca/~cl/software/COBA/coba2.html>
- ▶ Protege
 - ▶ Ontology editing
 - ▶ <http://protege.stanford.edu/>
- ▶ General sources for open source software
 - ▶ Machine Learning: <http://mloss.org>
 - ▶ Argumentation: <http://argumentationcompetition.org/2015/solvers.html>
 - ▶ SAT: <http://satcompetition.org>

- 1 Introduction
- 2 Installation and Usage
 - Installation
 - Package overview
 - The implementation methodology behind Tweety
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- 1 Introduction
- 2 Installation and Usage
 - Installation
 - Package overview
 - The implementation methodology behind Tweety
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

Installation in 30 seconds 1/4

Terminal-only, all Linux derivatives/Mac OS X (below for Ubuntu), Java SDK 8 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 Tweety as dependency

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

3. Add Tweety as dependency

```
<project ...>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>net.sf.tweety</groupId>
      <artifactId>tweety-full</artifactId>
      <version>1.3</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 ) {  
        PropositionalFormula helloWorld =  
            new Proposition("HelloWorld");  
        System.out.println(helloWorld);  
    }  
}
```

6. Compile and run

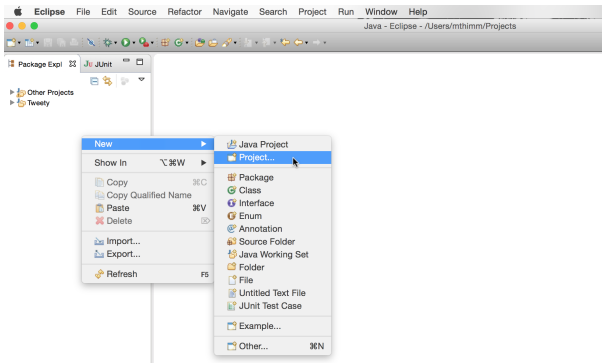
```
$ mvn package  
$ java -cp  
  target/mytweetyapp-1.0-SNAPSHOT-jar-with-dependencies.jar  
  mytweety.mytweetyapp.App  
HelloWorld
```

Installation in Eclipse 1/10

Prerequisites:

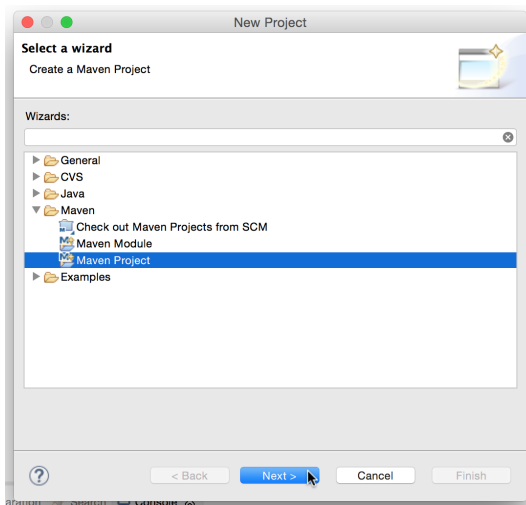
- ▶ Eclipse (<http://eclipse.org>)
- ▶ m2e plugin for Eclipse (<http://eclipse.org/m2e/>)

1. Create new project



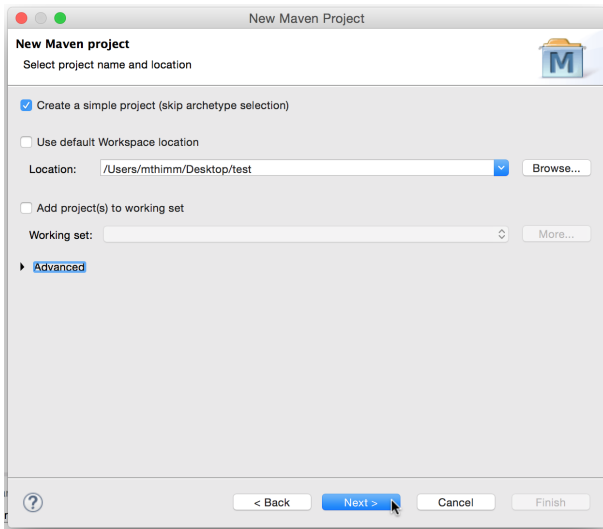
Installation in Eclipse 2/10

2. Select Maven Project

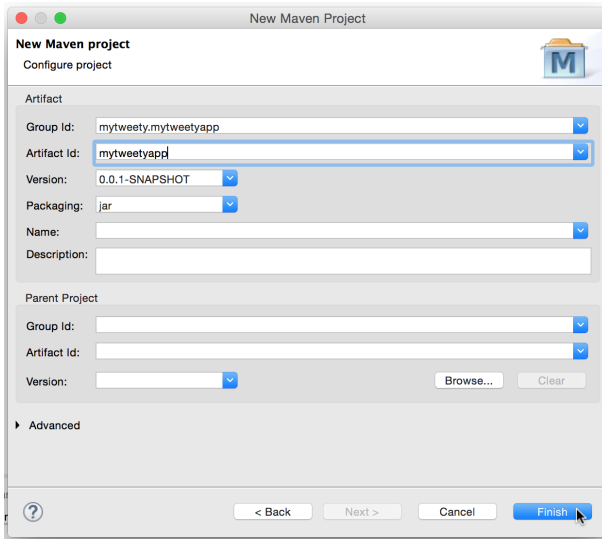


Installation in Eclipse 3/10

3. Configure Maven Project 1/2



4. Configure Maven Project 2/2

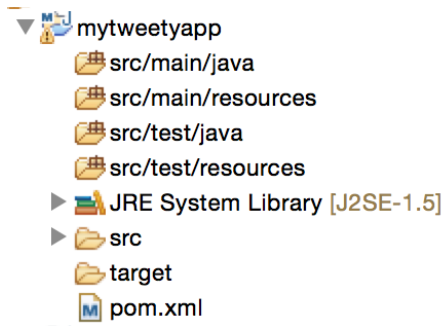


The screenshot shows the 'New Maven Project' dialog box in Eclipse IDE. The dialog is titled 'New Maven Project' and has a subtitle 'Configure project'. It features a Maven logo in the top right corner. The dialog is divided into several sections:

- Artifact:** This section contains five fields:
 - Group Id: mytweety.mytweetyapp
 - Artifact Id: mytweetyapp
 - Version: 0.0.1-SNAPSHOT
 - Packaging: jar
 - Name: (empty)
 - Description: (empty)
- Parent Project:** This section contains three fields:
 - Group Id: (empty)
 - Artifact Id: (empty)
 - Version: (empty)There are 'Browse...' and 'Clear' buttons next to the Version field.
- Advanced:** This section is currently collapsed, indicated by a right-pointing triangle.

At the bottom of the dialog, there are four buttons: '?', '< Back', 'Next >', and 'Finish'. The 'Finish' button is highlighted with a mouse cursor.

5. Overview on new project

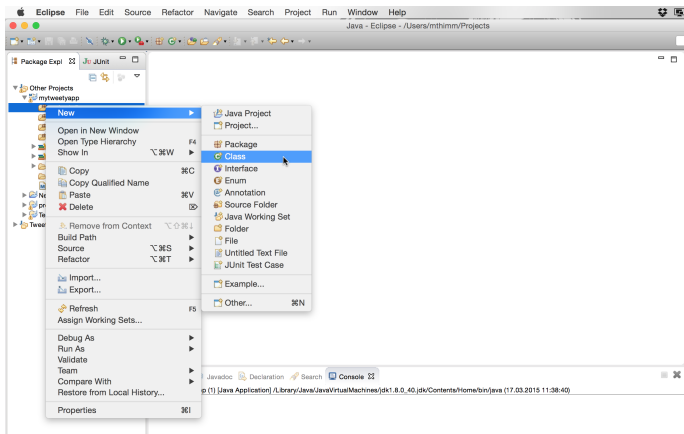


Installation in Eclipse 6/10

6. Add Tweety dependency to pom.xml

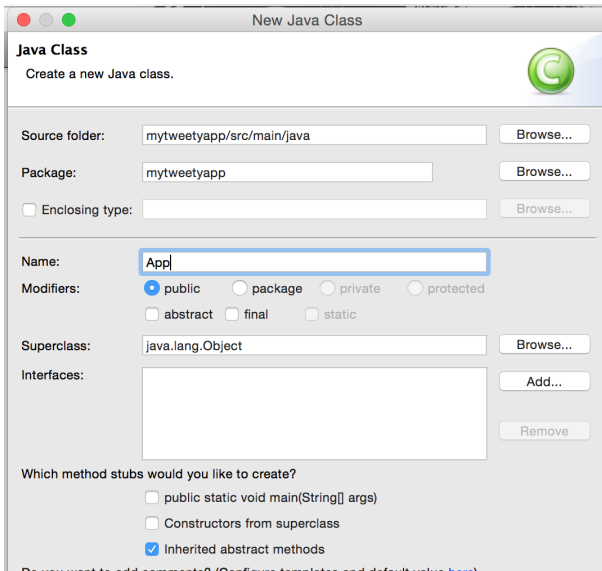
```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>mytweety</groupId>
4 <artifactId>mytweetyapp</artifactId>
5 <version>0.0.1-SNAPSHOT</version>
6 <dependencies>
7 <dependency>
8 <groupId>net.sf.tweety</groupId>
9 <artifactId>tweety-full</artifactId>
10 <version>1.3</version>
11 </dependency>
12 </dependencies>
13 </project>
```


7. Create new source file



Installation in Eclipse 8/10

7. Create new source file



The screenshot shows the 'New Java Class' dialog box in Eclipse. The title bar reads 'New Java Class'. The main heading is 'Java Class' with a sub-instruction 'Create a new Java class.' and a green circular icon with a 'C'.

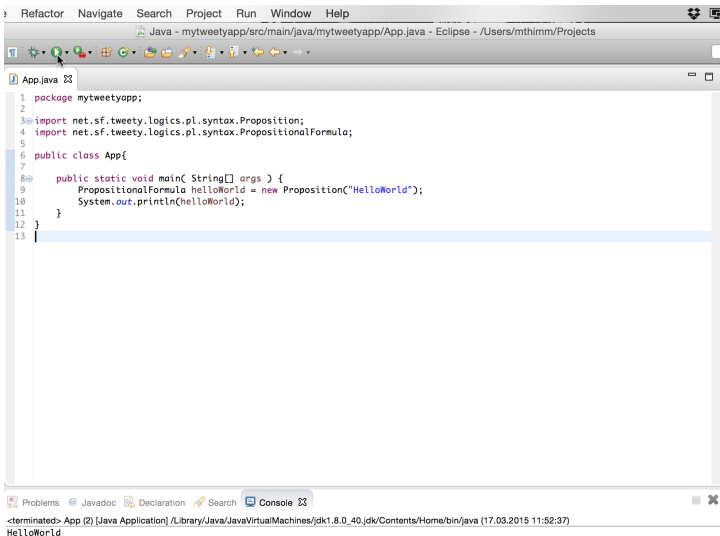
Fields and options include:

- Source folder: mytweetyapp/src/main/java (with a 'Browse...' button)
- Package: mytweetyapp (with a 'Browse...' button)
- Enclosing type: (checkbox) (with a 'Browse...' button)
- Name: App (highlighted with a blue border)
- Modifiers: public, package, private, protected, abstract, final, static
- Superclass: java.lang.Object (with a 'Browse...' button)
- Interfaces: (empty list with 'Add...' and 'Remove' buttons)
- Which method stubs would you like to create?
 - public static void main(String[] args)
 - Constructors from superclass
 - Inherited abstract methods
- Do you want to add comments? (Configure templates and default value here)

8. Write code

```
App.java ⌵
1 package mytweetyapp;
2
3 import net.sf.tweety.logics.pl.syntax.Proposition;
4 import net.sf.tweety.logics.pl.syntax.PropositionalFormula;
5
6 public class App{
7
8     public static void main( String[] args ) {
9         PropositionalFormula helloWorld = new Proposition("HelloWorld");
10        System.out.println(helloWorld);
11    }
12 }
13
```

9. Run



The screenshot shows the Eclipse IDE interface. The top menu bar includes Refactor, Navigate, Search, Project, Run, Window, and Help. The title bar indicates the current file is 'App.java' in the project 'mytweetapp'. The toolbar contains various icons, including a green play button (Run) which is highlighted by a mouse cursor. The main editor window displays the following Java code:

```
1 package mytweetapp;
2
3 import net.sf.tweety.logics.pl.syntax.Proposition;
4 import net.sf.tweety.logics.pl.syntax.PropositionalFormula;
5
6 public class App{
7
8     public static void main( String[] args ) {
9         PropositionalFormula helloWorld = new Proposition("HelloWorld");
10        System.out.println(helloWorld);
11    }
12 }
13 |
```

At the bottom of the IDE, the Console view is active, showing the output of the program:

```
<terminated> App [2] [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin/java (17.03.2015 11:52:37)
HelloWorld
```

Using the snapshot versions of Tweety 1/8

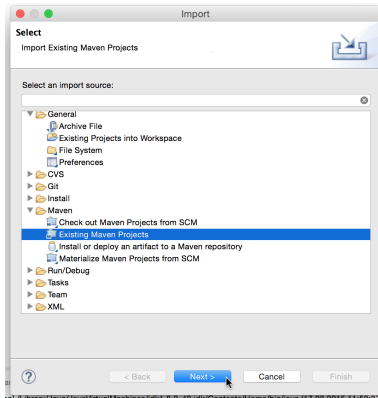
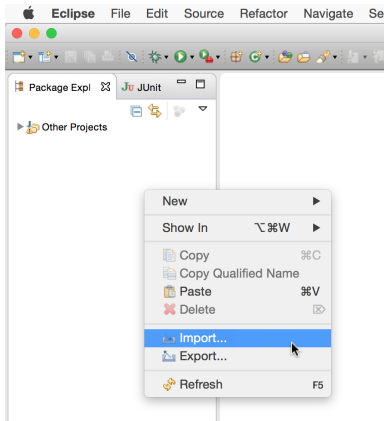
- ▶ The previous instructions always used the latest Maven version of Tweety (updated about twice a year)
- ▶ In order to use the most up-to-date version we recommend using the snapshot versions from SVN with Eclipse

1. Retrieve current version from SVN

```
$ svn checkout svn://svn.code.sf.net/p/tweety/code/ tweety-code
```

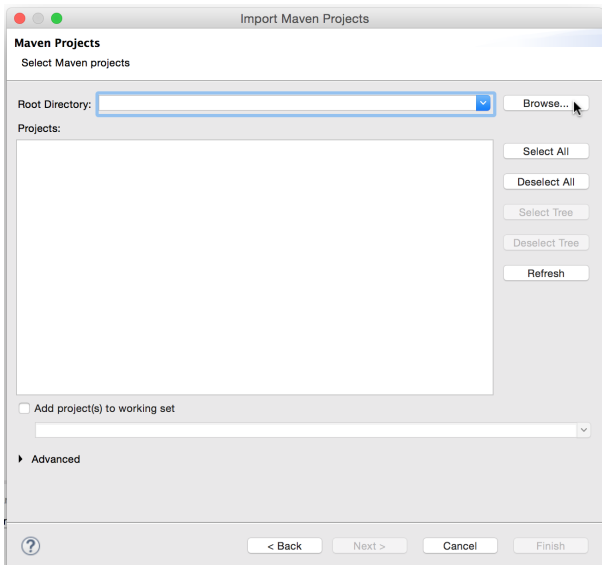
Using the snapshot versions of Tweety 2/8

2. Import into Eclipse



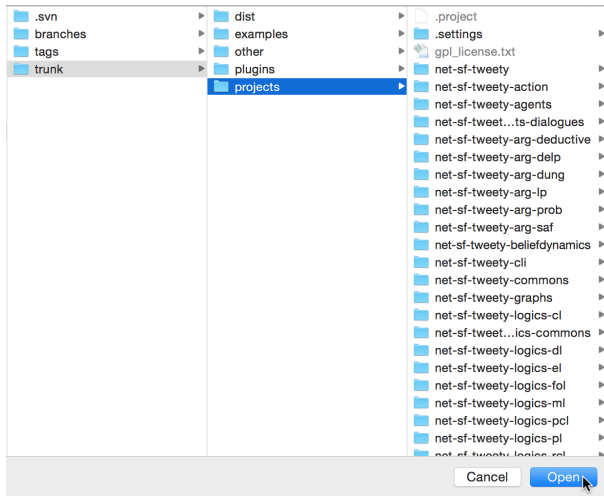
Using the snapshot versions of Tweety 3/8

3. Search for checked-out local copy



Using the snapshot versions of Tweety 4/8

4. Select “projects” folder of SVN local copy



Using the snapshot versions of Tweety 5/8

5. Configure options

Import Maven Projects

Maven Projects
Select Maven projects

Root Directory:

Projects:

- /pom.xml net.sf.tweety:parent-pom:1.3-SNAPSHOT:pom
- net-sf-tweety-commons/pom.xml net.sf.tweety:commons:1.3-SNAPSHOT:jar
- net-sf-tweety-math/pom.xml net.sf.tweety:math:1.3-SNAPSHOT:jar
- net-sf-tweety-action/pom.xml net.sf.tweety:action:1.3-SNAPSHOT:jar
- net-sf-tweety-beliefdynamics/pom.xml net.sf.tweety:beliefdynamics:1.3-SNAPSHOT:jar
- net-sf-tweety-clj/pom.xml net.sf.tweety:clj:1.3-SNAPSHOT:jar
- net-sf-tweety-graphs/pom.xml net.sf.tweety:graphs:1.3-SNAPSHOT:jar
- net-sf-tweety-preferences/pom.xml net.sf.tweety:preferences:1.3-SNAPSHOT:jar
- net-sf-tweety-machinelearning/pom.xml net.sf.tweety:machinelearning:1.3-SNAPSHOT:jar
- net-sf-tweety-plugin/pom.xml net.sf.tweety:plugin:1.3-SNAPSHOT:jar
- net-sf-tweety-ip-asp/pom.xml net.sf.tweety:ip:asp:1.3-SNAPSHOT:jar
- net-sf-tweety-ip-asp-beliefdynamics/pom.xml net.sf.tweety:ip:asp:beliefdynamics:1.3-SNAPSHOT:jar
- net-sf-tweety-ip-nlp/pom.xml net.sf.tweety:ip:nlp:1.3-SNAPSHOT:jar
- net-sf-tweety-logics-commons/pom.xml net.sf.tweety:logics:commons:1.3-SNAPSHOT:jar
- net-sf-tweety-logics-cl/pom.xml net.sf.tweety:logics:cl:1.3-SNAPSHOT:jar
- net-sf-tweety-logics-fof/pom.xml net.sf.tweety:logics:fof:1.3-SNAPSHOT:jar
- net-sf-tweety-logics-pl/pom.xml net.sf.tweety:logics:pl:1.3-SNAPSHOT:jar
- net-sf-tweety-logics-ml/pom.xml net.sf.tweety:logics:ml:1.3-SNAPSHOT:jar

Add project(s) to working set

Advanced

- Resolve Workspace projects

Profiles:

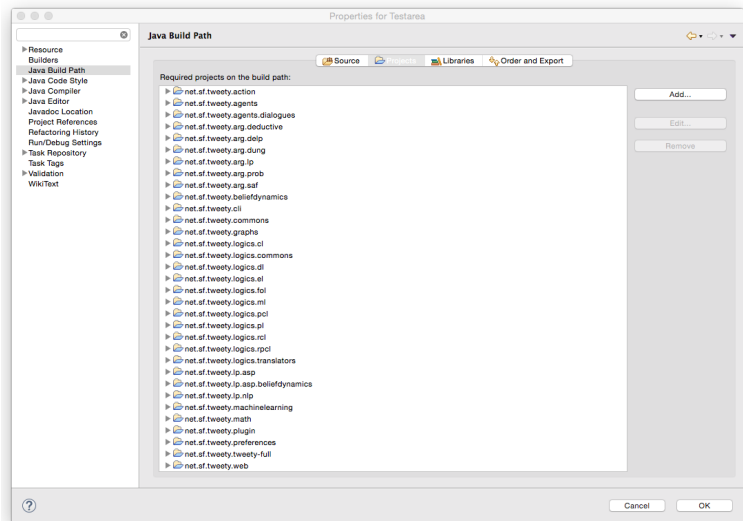
Name template:

6. All Tweety libraries in one working set

- ▼ Tweety
 - ▶ net.sf.tweety.action
 - ▶ net.sf.tweety.agents
 - ▶ net.sf.tweety.agents.dialogues
 - ▶ net.sf.tweety.arg.deductive
 - ▶ net.sf.tweety.arg.delp
 - ▶ net.sf.tweety.arg.dung
 - ▶ net.sf.tweety.arg.lp
 - ▶ net.sf.tweety.arg.prob
 - ▶ net.sf.tweety.arg.saf
 - ▶ net.sf.tweety.beliefdynamics
 - ▶ net.sf.tweety.cli
 - ▶ net.sf.tweety.common
 - ▶ net.sf.tweety.graphs
 - ▶ net.sf.tweety.logics.cl
 - ▶ net.sf.tweety.logics.common
 - ▶ net.sf.tweety.logics.dl
 - ▶ net.sf.tweety.logics.el
 - ▶ net.sf.tweety.logics.fol
 - ▶ net.sf.tweety.logics.ml
 - ▶ net.sf.tweety.logics.pcl
 - ▶ net.sf.tweety.logics.pl
 - ▶ net.sf.tweety.logics.rcl
 - ▶ net.sf.tweety.logics.rpcl
 - ▶ net.sf.tweety.logics.translators
 - ▶ net.sf.tweety.lp.asp
 - ▶ net.sf.tweety.lp.asp.beliefdynamics
 - ▶ net.sf.tweety.lp.nlp
 - ▶ net.sf.tweety.machinelearning
 - ▶ net.sf.tweety.math
 - ▶ net.sf.tweety.parent-pom
 - ▶ net.sf.tweety.plugin
 - ▶ net.sf.tweety.preferences
 - ▶ net.sf.tweety.tweety-full
 - ▶ net.sf.tweety.web

Using the snapshot versions of Tweety 7/8

7. Add libraries to other projects in Eclipse



8. Update regularly to get the most recent version

```
$ svn up
```

Contributing to Tweety

- ▶ Tweety 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@mthimm.de)
- ▶ Write-access to the repository will be enabled afterwards

- 1 Introduction
- 2 Installation and Usage
 - Installation
 - Package overview
 - The implementation methodology behind Tweety
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

General Libraries:

- ▶ Tweety Commons (`net.sf.tweety.common`)
- ▶ Command Line Interface (`net.sf.tweety.cli`)
- ▶ Plugin (`net.sf.tweety.plugin`)
- ▶ Math (`net.sf.tweety.math`)
- ▶ Graphs (`net.sf.tweety.graphs`)
- ▶ Web (`net.sf.tweety.web`)

Logic Libraries

- ▶ Logic Commons (`net.sf.tweety.logics.common`)
- ▶ 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`)
- ▶ Relational Probabilistic Conditional Logic
(`net.sf.tweety.logics.rpcl`)
- ▶ Markov Logic (`net.sf.tweety.logics.ml`)
- ▶ Epistemic Logic (`net.sf.tweety.logics.el`)
- ▶ Description Logic (`net.sf.tweety.logics.pl`)
- ▶ Logic Translators (`net.sf.tweety.logics.translators`)

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`)

Argumentation Libraries:

- ▶ Abstract Argumentation (`net.sf.tweety.arg.dung`)
- ▶ Deductive Argumentation (`net.sf.tweety.arg.deductive`)
- ▶ 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`)

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`)

- 1 Introduction
- 2 Installation and Usage
 - Installation
 - Package overview
 - The implementation methodology behind Tweety
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ Tweety 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, ...)
- ▶ Tweety 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

- ▶ 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
- ▶ Tweety provides lots of general applicable mathematical and logical background to easily implement even sophisticated KR formalisms

- ▶ Tweety provides propriety implementations of many KR *languages* (starting from propositional logic)
- ▶ Tweety 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
 - ▶ Tweety 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

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts**
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ In order to implement a new KR formalism, usually the following interfaces have to be implemented
 1. `Formula`: what constitutes the syntactic structures of the formalism?
 2. `BeliefBase`: how are formulas organized in a belief base (as a set, ordered list, ...)?
 3. `Interpretation`: how is the syntax interpreted?
 4. `Reasoner`: how do formulas and belief bases entail further formulas?
- ▶ Further interfaces/abstract classes can help
 1. `Signature`: what are the basic ingredients of the syntax?
 2. `RelationalFormula`: formulas based on first-order logic (contains utilities for grounding, variable substitution, ...)
 3. `Conjunctable`, `Disjunctable`: for formulas that can be combined using e.g. conjunction and disjunction
 4. ...

The basic classes/interfaces are “very” abstract at the top level

```
public interface Formula{
    public Signature getSignature();
}
```

```
public interface BeliefBase {
    public Signature getSignature();
    public String toString();
}
```

```
public interface Interpretation {
    public boolean satisfies(Formula formula);
    public boolean satisfies
        (Collection<? extends Formula> formulas);
    public boolean satisfies(BeliefBase beliefBase);
}
```

Basic classes 3/3

```
public abstract class Reasoner {
    private BeliefBase beliefBase;
    public Reasoner(BeliefBase beliefBase){
        this.beliefBase = beliefBase;
    }
    public abstract Answer query(Formula query);
    public BeliefBase getKnowledgBase(){
        return this.beliefBase;
    }
}

public abstract class Signature {
    public abstract boolean isSubSignature
        (Signature other);
    public abstract boolean isOverlappingSignature
        (Signature other);
    public abstract void addSignature
        (Signature other);
}
```

Parser and Writer should be provided to loading and saving knowledge bases to the disk:

```
public abstract class Parser<T extends BeliefBase> {
    public T parseBeliefBaseFromFile(String filename)
        { ... }
    public T parseBeliefBase(String text) { ... }
    public abstract T parseBeliefBase(Reader reader);
    public Formula parseFormulaFromFile
        (String filename) { ... }
    public Formula parseFormula(String text) { ... }
    public abstract Formula parseFormula
        (Reader reader);
}
```

```
public abstract class Writer {  
    private Object obj;  
    public Writer(Object obj) { ... }  
    public void setObject(Object obj) { ... }  
    public Object getObject() { ... }  
    public abstract String writeToString();  
    public void writeToFile(String filename) { ... }  
}
```

- ▶ If you implemented all classes mentioned before properly, you can create a Tweety plugin and get a command line interface for free
 - ▶ Builds on JSPF (Java Simple Plugin Framework)
 - ▶ Encapsulates language-independent functionalities in a general CLI
- ▶ Also in development: a general Tweety REST API
 - ▶ `net.sf.tweety.web`
 - ▶ Case study on inconsistency management
<http://tweetyproject.org/w/incmes>

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic**
 - Using and reasoning with propositional logic
 - Using SAT solvers
 - Exercises
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

Definition

A *propositional signature* At is a finite set of identifiers, called *atoms* or *propositions*.

Definition

Let At be a propositional signature. The *propositional language* $\mathcal{L}(At)$ for At is the minimal set \mathcal{L} satisfying

1. $At \subseteq \mathcal{L}$,
2. $\top, \perp \in \mathcal{L}$ (tautology and contradiction), and
3. for every $\phi, \psi \in \mathcal{L}$ it holds that
 - 3.1 $\phi \wedge \psi \in \mathcal{L}$ (conjunction),
 - 3.2 $\phi \vee \psi \in \mathcal{L}$ (disjunction), and
 - 3.3 $\neg\phi \in \mathcal{L}$ (negation).

Propositional Logic - Recap 2/4

Definition

Let At be a propositional signature. A *propositional interpretation* I on At is a function

$$I : At \rightarrow \{\text{true}, \text{false}\} \quad .$$

Let $\text{Int}(At)$ denote the set of all propositional interpretations for At . An interpretation can also be written as a *complete conjunction* enumerating all literals that are true in the given interpretation.

Example

Consider a propositional signature $At = \{a, b, c\}$. The interpretation I_1 of At given by

$$I_1(a) = \text{true} \qquad I_1(b) = \text{false} \qquad I_1(c) = \text{true}$$

can be fully described by the complete conjunction $a\bar{b}c$.

Propositional Logic - Recap 3/4

- ▶ An interpretation I *satisfies* an atom $a \in \text{At}$, denoted by $I \models^P a$, if and only if $I(a) = \text{true}$.
- ▶ An interpretation I *falsifies* an atom $a \in \text{At}$, denoted by $I \not\models^P a$, if and only if $I(a) = \text{false}$.

The satisfaction relation \models^P is extended to arbitrary sentences recursively as follows. Let $\phi, \psi \in \mathcal{L}(\text{At})$ be some sentences.

- ▶ $I \models^P \phi \vee \psi$ if and only if $I \models^P \phi$ or $I \models^P \psi$
- ▶ $I \models^P \phi \wedge \psi$ if and only if $I \models^P \phi$ and $I \models^P \psi$
- ▶ $I \models^P \neg\phi$ if and only if $I \not\models^P \phi$

Furthermore, for every interpretation I it holds that $I \models^P \top$ and $I \not\models^P \perp$.

- ▶ I is a *propositional model* of a sentence $\phi \in \mathcal{L}(\text{At})$ if and only if $I \models^P \phi$.
- ▶ Let $\text{Mod}^P(\Phi) \subseteq \text{Int}(\text{At})$ denote the set of all models of $\Phi \subseteq \mathcal{L}(\text{At})$.
- ▶ A set of formulas Φ_2 *semantically follows* from a set of formulas Φ_1 , denoted by $\Phi_1 \models^P \Phi_2$, if and only if $\text{Mod}^P(\Phi_1) \subseteq \text{Mod}^P(\Phi_2)$.

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic**
 - Using and reasoning with propositional logic
 - Using SAT solvers
 - Exercises
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

Every concept is mapped 1:1 in Tweety

- ▶ `public abstract class PropositionalFormula`
- ▶ `public class Proposition extends PropositionalFormula`
- ▶ `public class Negation extends PropositionalFormula`
- ▶ `public class Conjunction extends PropositionalFormula`
- ▶ ...
- ▶ `public class PropositionalSignature extends SetSignature<Proposition>`
- ▶ `public class PlBeliefSet extends BeliefSet<PropositionalFormula>`
- ▶ `public class PossibleWorld extends InterpretationSet<Proposition>`

PropositionalFormula (Excerpt)

```
public abstract class PropositionalFormula
    implements ClassicalFormula {
    PropositionalSignature getSignature() { ... }
    abstract Set<Proposition> getAtoms();
    abstract Set<PropositionalFormula> getLiterals();
    Conjunction combineWithAnd(Conjunctable f) { ... }
    Disjunction combineWithOr(Disjunctable f) { ... }
    abstract PropositionalFormula
        collapseAssociativeFormulas();
    abstract PropositionalFormula trim() { ... }
    abstract PropositionalFormula toNnf();
    abstract Conjunction toCnf();
    Set<PossibleWorld> getModels() { ... }
    PropositionalFormula toDnf() { ... }
    ClassicalFormula complement() { ... }
}
```

Constructing formulas

```
Proposition p = new Proposition("p");  
Proposition q = new Proposition("q");  
  
PropositionalFormula f1 = new Conjunction(p,q);  
  
PropositionalFormula f2 = p.combineWithAnd(q);  
  
PropositionalFormula f3 =  
    p.combineWithAnd(new Negation(q)).combineWithOr(q);
```

Tweety file format for propositional logic

```
p  
p && q  
r || !s  
p && (!s || !q)
```

```
PlParser parser = new PlParser();  
PlBeliefSet f3 = parser.parseBeliefBaseFromFile(file);
```

Interpretations: Possible Worlds

Semantics is mapped 1:1 in possible worlds:

```
public class PossibleWorld extends ... {
    private Set<Proposition> truePropositions;
    public boolean satisfies(Formula formula){
        ...
        if(formula instanceof Contradiction)
            return false;
        if(formula instanceof Tautology)
            return true;
        if(formula instanceof Proposition)
            return this.contains(formula);
        if(formula instanceof Negation)
            return !this.satisfies(((Negation)formula)
                .getFormula());
        if(formula instanceof Conjunction){
            Conjunction c = (Conjunction) formula;
            for(PropositionalFormula f : c)
                if(!this.satisfies(f))
                    return false;
            return true;
        } ... }
}
```


- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic**
 - Using and reasoning with propositional logic
 - **Using SAT solvers**
 - Exercises
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

Reasoning with Propositional Logic

- ▶ Having a semantical characterization of propositional logic *within* the programming environment is good for verification purposes
- ▶ For practical reasoning, however, one should use existing SAT solvers
- ▶ Tweety has built-in support for various SAT solvers
 - ▶ Sat4J: Java-based SAT solver (no additional libraries or external executables needed)
 - ▶ Lingeling (external binary compiled for system needed)
 - ▶ basically any SAT solver with a command line interface compatible with the SAT competition requirements
- ▶ Basic approach: an inference problem is reduced to a consistency problem (note that $\Phi \models^P \alpha$ if and only if $\Phi \cup \{\neg\alpha\}$ is inconsistent)

SAT solvers in Tweety 1/2

```
SatSolver mySolver = new Sat4jSolver();  
  
PlBeliefSet kb = ...  
  
System.out.println(mySolver.isConsistent(kb));
```

SAT solvers in Tweety 2/2

SAT solvers are managed with static methods in `SatSolver`

```
SatSolver.setDefaultSolver  
    (new LingelingSolver("path/to/binary"));
```

Now Lingeling is used as the default solver for everything

```
PlBeliefSet kb = ...  
PropositionalFormula queryFormula = ...  
  
SatReasoner reasoner = new SatReasoner(kb);  
  
System.out.println(reasoner.query(queryFormula));
```

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic**
 - Using and reasoning with propositional logic
 - Using SAT solvers
 - Exercises
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

E1. Create and print propositional formulas

Create representations of the following propositional formulas:

1. $a \vee (b \wedge c)$
2. $a \wedge \neg a$
3. $\neg(\neg\neg a \wedge \neg b)$
4. $a \vee b \vee c \vee \neg d$

Afterwards print each formula on the terminal.

E2. Formula manipulation and CNF

1. Create a disjunction of the four formulas from the previous exercise.
2. Print the formula on the terminal
3. Convert the formula into CNF
4. Print the converted formula on the terminal

E3. Satisfiability

Create a representation of the following propositional formula using `PlParser`:

$$(b \vee \neg d) \wedge (\neg a \vee (b \wedge \neg(c \vee d) \wedge e) \vee (a \wedge \neg c))$$

1. Print the formula on the terminal
2. Is this formula satisfiable? Try out `Sat4jSolver`
3. Print a model of the formula

E4. Interpretations

1. Create a propositional signature with 4 propositions $\{A0, A1, A2, A3\}$
2. Use `PossibleWorldIterator` to print all interpretations of the corresponding propositional language
3. Create a representation of the formula $A1 \vee (A2 \wedge \neg A3)$
4. Use `PossibleWorldIterator` to print all interpretations that satisfy the formula from 3.

E5. Sampling belief bases

Try out sampling belief bases using `CnfSampler`

1. Create a propositional signature with 10 propositions
2. Create a `CnfSampler` for this signature with a variable ratio of 0.3
3. Sample 10 belief bases with 5–10 formulas each

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs**
 - Mathematical tools in Tweety
 - Constraint Satisfaction and Optimization problems
 - Using general graph structures
 - Exercises
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ Many approaches in KR and AI in general rely on mathematical formalisms
- ▶ This includes
 - ▶ Set theory, set operations
 - ▶ mathematical terms, aggregation of mathematical terms
 - ▶ constraint satisfaction, optimization problems
 - ▶ matrices, vectors
 - ▶ probability theory
- ▶ The package `net.sf.tweet.math` provides helpful utilities for dealing with mathematical subproblems
- ▶ The package `net.sf.tweet.graphs` provides methods for working with graphs

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
 - Mathematical tools in Tweety
 - Constraint Satisfaction and Optimization problems
 - Using general graph structures
 - Exercises
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ Strictly speaking, the mathematical language of terms, equations, ... is also a KR language
- ▶ Tweety uses the same methodology for representing mathematical terms as for its KR approaches
- ▶ The syntax of mathematical terms:
 - ▶ `Term` as abstract ancestor for all terms; `IntegerConstant` as atomic element
 - ▶ `Product`, `Fraction`, `Sum` as connectors
 - ▶ Representations of functions: `Logarithm`, `Exp`, `Root`, ...
- ▶ `Equation` as the basic formula (consisting of two terms)

Example

The equation

$$X + 2Y = 3Z - 5$$

can be represented via

```
Variable x = new IntegerVariable("X");  
Variable y = new IntegerVariable("Y");  
Variable z = new IntegerVariable("Z");  
Constant two = new IntegerConstant(2);  
Constant three = new IntegerConstant(3);  
Constant five = new IntegerConstant(5);
```

```
Equation eq = new Equation(  
    x.add(two.mult(y)),  
    three.mult(z).minus(five)  
);
```

Working with mathematical terms (cont'd)

- ▶ Tweety provides several tools for working with terms
 - ▶ Automatic derivation ($3X^2 \rightarrow 6X$)
 - ▶ Checking whether a function is continuous
 - ▶ Bringing terms in normal form
- ▶ Representations of vectors and matrices
- ▶ Matrix multiplication and other algebraic operations
- ▶ One important aspect are constraint satisfaction and optimization problems

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs**
 - Mathematical tools in Tweety
 - Constraint Satisfaction and Optimization problems**
 - Using general graph structures
 - Exercises
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ Many reasoning approaches which involve quantitative uncertainty (probabilistic logics, fuzzy logics, etc.) or search problems involve optimization problems
 - ▶ Compute a probability function with maximum entropy
 - ▶ Find a shortest path
 - ▶ Paraconsistent reasoning: minimize number of propositions receiving a paraconsistent truth value
- ▶ The package `net.sf.tweety.math.opt` provides general classes for modelling optimization problems (and constraint satisfaction problems) and bridges to methods solving those

Optimization Problems

An optimization problem P is of the form

$$\begin{aligned} &\text{Maximize} && F(X_1, \dots, X_n) \\ &\text{subject to} && H_1(X_1, \dots, X_n) \leq B_1 \\ & && \dots \\ & && H_m(X_1, \dots, X_n) \leq B_m \end{aligned}$$

(or “Minimize”, “=”, “ \geq ”, ...)

- ▶ The class `OptimizationProblem` captures this definition
- ▶ Similarly, `ConstraintSatisfactionProblem` for constraint satisfaction problems
- ▶ The package `net.sf.tweety.math.opt.solver` contains implementations and bridges to several solvers

Solving Optimization Problems - Example

```
// minimize X+Y subject to X-Y >= 10 and Y >= 0
Solver.setDefaultLinearSolver(new ApacheCommonsSimplex());
OptimizationProblem problem =
    new OptimizationProblem(OptimizationProblem.MINIMIZE);
FloatVariable x = new FloatVariable("X");
FloatVariable y = new FloatVariable("Y");
problem.add(new Inequation(x.minus(y),
    new FloatConstant(10), Inequation.GREATER_EQUAL));
problem.add(new Inequation(y, new FloatConstant(0),
    Inequation.GREATER_EQUAL));
problem.setTargetFunction(x.add(y));

Map<Variable, Term> solution =
    Solver.getDefaultLinearSolver().solve(problem);
```

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
 - Mathematical tools in Tweety
 - Constraint Satisfaction and Optimization problems
 - Using general graph structures
 - Exercises
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ The package `net.sf.tweepy.graphs` contains representations of graphs
- ▶ General interface: `interface Graph<T extends Node> extends Iterable<T>`
- ▶ Representations of directed, undirected, and weighted edges
- ▶ Methods for determining Eigenvalues, strongly connected components, and others
- ▶ is currently being heavily extended (bridges to graph databases, layout components, etc.)

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs**
 - Mathematical tools in Tweety
 - Constraint Satisfaction and Optimization problems
 - Using general graph structures
 - Exercises
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

E6. Create and print mathematical statements

Create representations of the following mathematical statements:

1. $3 * X + 2X^2 = 44$
2. $\log X + \frac{X}{\log Y} > 0$
3. $e^{X-Y} \leq \sqrt[3]{X}$
4. $|X * Y| + \max\{0, X\} < 47$

Afterwards print each statement on the terminal.

E7. Working with mathematical statements

Create a representation of the following mathematical function (use float terms):

$$-9X + 2X^2 + \frac{2}{3}X^3$$

1. Print the term on the terminal
2. Print the derivation of the term wrt. X , why does it look like that?
3. Use the `simplify` method to simplify the derivation and print it again
4. Determine and print a root of the computed derivation (use `NewtonRootFinder` with starting point $X = 0$)

E8. Linear optimization

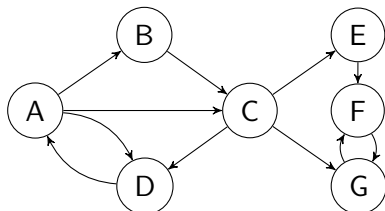
Solve the following linear optimization problem:

$$\begin{array}{ll} \text{Maximize} & 3X + 2Y \\ \text{subject to} & 2X + 4Y \leq 200 \\ & Y \geq 10 \end{array}$$

1. Create a representation of the above problem and print it on the terminal
2. Install the GLPK solver (e.g. via `sudo apt-get install glpk`)
3. Register GLPK as the default linear solver in Tweety (do not forget to provide correct path name and folder for temporary files)
4. Solve the above problem with the default linear solver, print the solution (variable assignments) and value of the solution

E9. Graphs

Consider the following directed graph:



1. Create a representation of the above graph and print it on the terminal (use `SimpleNode` for vertices)
2. What are the strongly connected components of this graph?
3. Print the adjacency matrix of the graph
4. What are the Eigenvalues of the graph?

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation**
 - Introduction to Computational Argumentation
 - Computational Argumentation in Tweety
 - Case Study: Strategic Argumentation
 - Exercises
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation**
 - **Introduction to Computational Argumentation**
 - Computational Argumentation in Tweety
 - Case Study: Strategic Argumentation
 - Exercises
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

Definition (Abstract Argumentation Framework)

An *abstract argumentation framework* AF is a tuple $AF = (\text{Arg}, \rightarrow)$ 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 $\mathcal{C} \rightarrow \mathcal{B}$

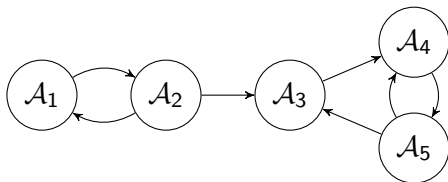
and it is *complete* if additionally

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

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.

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. $\Psi \not\vdash \perp$
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} .

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 $\alpha = \neg(\phi_1 \wedge \dots \wedge \phi_n)$ for some $\phi_1, \dots, \phi_n \subseteq \Phi$.

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

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

Remark: “undercut” in item 3.) should be “canonical undercut”

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 $\gamma : \mathcal{T}(\text{At}) \rightarrow \mathbb{R}$.

Definition

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

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

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

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation**
 - Introduction to Computational Argumentation
 - Computational Argumentation in Tweety**
 - Case Study: Strategic Argumentation
 - Exercises
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ Abstract Argumentation (`net.sf.tweety.arg.dung`)
- ▶ Deductive Argumentation (`net.sf.tweety.arg.deductive`)
- ▶ 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`)

Also relevant:

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

Creating and manipulating argumentation frameworks:

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

Parsing files in APX format:

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

Abstract Argumentation 2/2

Computing extensions:

```
DungTheory at = ...
```

```
CompleteReasoner r = new CompleteReasoner(at);  
System.out.println(r.getExtensions());
```

- ▶ Supported semantics: complete, grounded, preferred, stable, semi-stable, CF2, stage, ideal
- ▶ Standard reasoner returns objects of type `Extension` (sets of arguments)
- ▶ Conversion to `Labeling` also possible
- ▶ Most reasoners are based on semantical definitions (some on SAT solvers and some exploit SCC structure)
- ▶ Planned: add bridge to support command line interface solvers from `probo` (the interface standard of the argumentation competition)

Deductive Argumentation

Creating and manipulating knowledge bases (as in propositional logic):

```
DeductiveKnowledgeBase kb = new DeductiveKnowledgeBase();  
PlParser parser = new PlParser();  
  
kb.add( (PropositionalFormula)  
    parser.parseFormula("(a || b) && !c && d"));
```

Getting arguments:

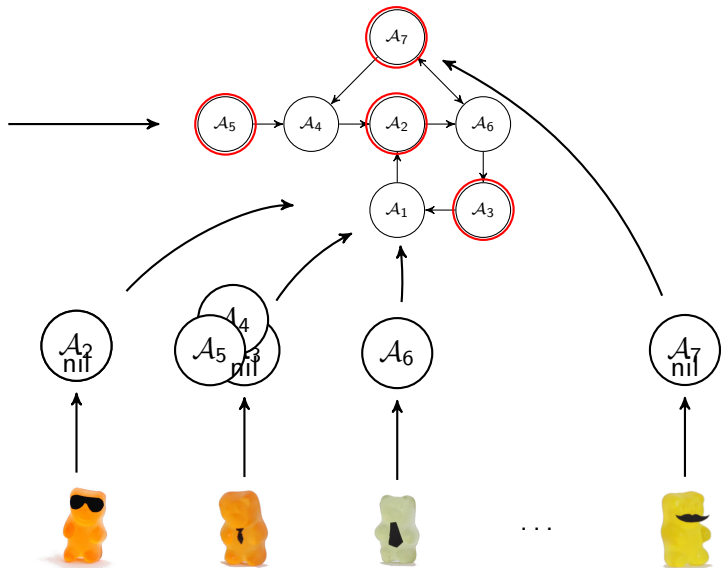
```
System.out.println(kb.getDeductiveArguments(someFormula));
```

Reasoning:

```
CompilationReasoner reasoner = new CompilationReasoner  
    (kb, new ClassicalCategorizer(),  
    new SimpleAccumulator());  
System.out.println(reasoner.query(someFormula)  
    .getAnswerBoolean());
```

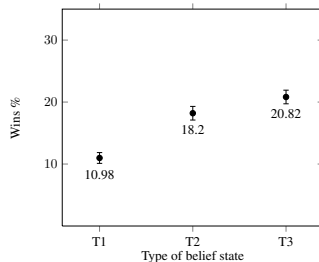
- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation**
 - Introduction to Computational Argumentation
 - Computational Argumentation in Tweety
 - Case Study: Strategic Argumentation**
 - Exercises
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

Strategic Argumentation



Empirical Evaluation with Tweety

- ▶ Implemented three different opponent models
- ▶ Implemented simulation environment for agent dialogues
 - ▶ Agent generators
 - ▶ Game generators
- ▶ Generated randomly 5000 argumentation frameworks
- ▶ Measured and compared performance of different models

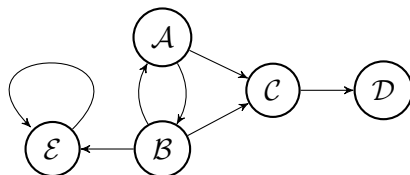


[Rienstra, Thimm, Oren. Opponent Models with Uncertainty for Strategic Argumentation. IJCAI 2013]

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation**
 - Introduction to Computational Argumentation
 - Computational Argumentation in Tweety
 - Case Study: Strategic Argumentation
 - Exercises
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

E10. Abstract Argumentation Frameworks

Consider the following abstract argumentation framework:



1. Create a representation of the above graph as an abstract argumentation framework
2. Compute and print out the complete, preferred, and stable extensions of the framework (do not forget to set your default SAT-solver)

E11. Abstract Argumentation Frameworks

1. Create an instance of `EnumeratingDungTheoryGenerator` and generate the first 100 abstract argumentation frameworks
2. Let us define the *stable extension number* as the number of stable extensions of an argumentation framework divided by the sum of the number of its arguments and attacks; print out this number for the 100 frameworks from above
3. Determine the *average stable extension number* for the considered frameworks (add up the numbers and divide by 100); increase the number of considered frameworks, does this number converge? To where and why?

E12. Deductive Argumentation

Consider the following propositional knowledge base \mathcal{K} :

$$\mathcal{K} = \{a, \neg a \vee b, \neg b, b \wedge \neg c, c \wedge a\}$$

1. Create a representation of the above knowledge as a deductive knowledge base
2. Compute and print out all deductive arguments for $a \vee b$
3. Is the formula a entailed by the knowledge base? Use `CompilationReasoner` with `ClassicalCategorizer` and `SimpleAccumulator`.

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement**
 - Introduction to Inconsistency Measurement
 - Inconsistency Measures in Tweety
 - Exercises
- 8 Summary and Conclusion

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement**
 - Introduction to Inconsistency Measurement
 - Inconsistency Measures in Tweety
 - Exercises
- 8 Summary and Conclusion

Inconsistency Measurement 1/3

Compare the following propositional knowledge bases

$$\mathcal{K}_1 = \{a, b \vee c, \neg a \wedge \neg b, d\} \quad \mathcal{K}_2 = \{a, \neg a, b, \neg b\}$$

- ▶ both \mathcal{K}_1 and \mathcal{K}_2 are inconsistent
- ▶ \mathcal{K}_2 seems *more* inconsistent than \mathcal{K}_1
- ▶ the field of *Inconsistency Measurement* is about quantifying inconsistency

Definition

A knowledge base \mathcal{K} is a finite set of propositional formulas. Let $\mathbb{K}(\text{At})$ be the set of all knowledge bases of signature At .

Definition

An *inconsistency measure* \mathcal{I} is a function $\mathcal{I} : \mathbb{K} \rightarrow [0, \infty)$.

Definition

A set $M \subseteq \mathcal{K}$ is called *minimal inconsistent subset* (MI) of \mathcal{K} if $M \models \perp$ and there is no $M' \subset M$ with $M' \models \perp$. Let $\text{MI}(\mathcal{K})$ be the set of all MIs of \mathcal{K} .

Definition

A formula $\alpha \in \mathcal{K}$ is called *free formula* of \mathcal{K} if there is no $M \in \text{MI}(\mathcal{K})$ with $\alpha \in M$. Let $\text{Free}(\mathcal{K})$ denote the set of all free formulas of \mathcal{K} .

Definition

A *basic inconsistency measure* is a function $\mathcal{I} : \mathbb{K} \rightarrow [0, \infty)$ that satisfies the following three conditions:

1. $\mathcal{I}(\mathcal{K}) = 0$ if and only if \mathcal{K} is consistent,
2. if $\mathcal{K} \subseteq \mathcal{K}'$ then $\mathcal{I}(\mathcal{K}) \leq \mathcal{I}(\mathcal{K}')$, and
3. for all $\alpha \in \text{Free}(\mathcal{K})$ we have $\mathcal{I}(\mathcal{K}) = \mathcal{I}(\mathcal{K} \setminus \{\alpha\})$.

Example

Define $\mathcal{I}_{\text{MI}}(\mathcal{K}) = |\text{MI}(\mathcal{K})|$

Then we have

$$\mathcal{I}_{\text{MI}}(\{a, \neg a, c\}) = 1$$

$$\mathcal{I}_{\text{MI}}(\{a, b, \neg a \wedge \neg b\}) = 2$$

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement**
 - Introduction to Inconsistency Measurement
 - **Inconsistency Measures in Tweety**
 - Exercises
- 8 Summary and Conclusion

Inconsistency Measures in Tweety 1/2

General interface:

```
interface InconsistencyMeasure<T extends BeliefBase> {  
    public Double inconsistencyMeasure(T beliefBase);  
}
```

Available implementations: \mathcal{I}_{MI} , \mathcal{I}_d , \mathcal{I}_{MI^c} , \mathcal{I}_η , \mathcal{I}_c , \mathcal{I}_{mc} , \mathcal{I}_p , \mathcal{I}_{hs} ,
 $\mathcal{I}_{dalal}^\Sigma$, $\mathcal{I}_{dalal}^{\max}$, $\mathcal{I}_{dalal}^{\text{hit}}$, \mathcal{I}_{D_f} , \mathcal{I}_{P_m} , \mathcal{I}_{mv} , \mathcal{I}_{nc} , ...

- ▶ Tool needed for many measures: MUS enumerators
- ▶ Available implementations of `PlMusEnumerator`:
 - ▶ `MimusMusEnumerator`
 - ▶ `MarcoMusEnumerator`
 - ▶ `NaiveMusEnumerator`

Inconsistency Measures in Tweety 2/2

Usage:

```
PlMusEnumerator.setDefaultEnumerator(  
    new NaiveMusEnumerator<PropositionalFormula>  
        (new Sat4jSolver()));  
  
MiInconsistencyMeasure<PropositionalFormula> miInc =  
    new MiInconsistencyMeasure<PropositionalFormula>  
        (PlMusEnumerator.getDefaultEnumerator());  
  
PlBeliefSet kb = ...  
System.out.println(miInc.inconsistencyMeasure(kb));
```




Inconsistency Measurement (IM) is a subfield of Knowledge Representation (KR) that deals with quantitative measures of inconsistency. Formally, an inconsistency measure \mathcal{I} is a function that assigns to any knowledge base \mathcal{K} in propositional logic a value $\mathcal{I}(\mathcal{K}) \in [0, \infty)$ with the idea that larger values indicate a larger inconsistency in \mathcal{K} . In recent years, a lot of proposals have been given of how an inconsistency measure \mathcal{I} can be defined. For a more elaborate introduction to inconsistency measures see the [Technical Documentation](#) and the references listed therein.

This page complements the theoretical research in IM by providing an experimentation platform for the practical use of inconsistency measures. It is part of the [Tweety collection of Java libraries for logical aspects of artificial intelligence and knowledge representation](#) which contains implementations of various inconsistency measures. This page serves as documentation to these implementations and provides both a web-based interface for using these implementations and a REST API for integrating them in your own applications.

Online Inconsistency Measurement

You can try out different inconsistency measures directly below by entering a knowledge base in propositional logic in the field *Knowledge base*. The syntax used on this page is the one of Boolean expressions in most programming languages (such as Java). Each line in the field is a separate formula and each formula can be composed using arbitrary propositions (starting with a letter or "_") and containing only letters, numbers, and "." and the connectives "&&" (conjunction), "||" (disjunction), and "!" (negation). The operators follow the usual order of preference ("!" "&&" ">" "||") and parentheses "(" and ")" can be used to override this order.

Once a knowledge base has been entered a set of inconsistency measures can be selected (using the button "...") and the inconsistency values of those measures wrt. the given knowledge base are computed using the button "Compute inconsistency values". The computation of a measure can be aborted at any time by pressing the corresponding button (note that for large knowledge bases the computation of some measures can take quite some time). The request is handled on the server and the runtime of a request on the server is also shown besides the actual inconsistency value.

For more information on the individual inconsistency measures see the [Technical Documentation](#).

Knowledge base:

```
A
B
A && !C
!A || C
!B || C
B && !C
(D || B) && !A
!D && (F || !C)
D && C
```

Selected inconsistency measures:



REST API and source code

The above web interface uses a REST API to remotely compute inconsistency values. If you need to integrate inconsistency measurement in your own applications you can also use this REST API directly. Communication between a client and the server is done using JSON (in UTF-8), the URL of the server is <http://141.26.208.49:8080/tweety/incmes>. The API supports two commands:

1. Get list of inconsistency measures

Simply send a JSON of the form

```
{
  "cmd": "measures",
  "email": "<your e-mail>"
}
```

to the server. The field "email" serves as an identification of your request. I recommend to use a valid e-mail address as it can also be used if issues arise during the computation and I would like to contact you. However, if you feel uncomfortable with that you may also use any other arbitrary identification string.

In reply to the above request the server will send you back a JSON of the form

```
{
  "measures":
  [
    {
      "id": "drastic",
      "label": "Drastic Inconsistency Measure",
      "description": "<some description in HTML>"
    },
    {
      "id": "mi",
      "label": "MI Inconsistency Measure",
      "description": "<some description in HTML>"
    },
    ...
  ]
  "reply": "measures",
  "email": "<your e-mail>"
}
```

which lists all currently available measures with their id.

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement**
 - Introduction to Inconsistency Measurement
 - Inconsistency Measures in Tweety
 - Exercises
- 8 Summary and Conclusion

E13. MUSes and MCSes

Consider the following propositional knowledge base \mathcal{K} :

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

1. Create a representation of the above knowledge base as a propositional belief set.
2. Configure `NaiveMusEnumerator` with `Sat4jSolver` as your default MUS enumerator for propositional logic
3. Use your default MUS enumerator for propositional logic to determine all MUSes and MCSes of \mathcal{K}

E14. Inconsistency Measures

Consider again the following propositional knowledge base \mathcal{K} :

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

1. The packages `net.sf.tweety.logics.common.analysis` and `net.sf.tweety.logics.pl.analysis` contain several inconsistency measures; select three of those and compute their inconsistency value of \mathcal{K} .

Outline

- 1 Introduction
- 2 Installation and Usage
- 3 Basics: Important classes and concepts
- 4 Basics: Propositional Logic
- 5 Basics: Mathematics and Graphs
- 6 Advanced topics: Computational Argumentation
- 7 Advanced topics: Inconsistency Measurement
- 8 Summary and Conclusion

- ▶ 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 analytically, but experimental evaluation helps for trial-and-error purposes

- ▶ Tweety 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@mthimm.de)
- ▶ Write-access to the repository will be enabled afterwards

Final Remarks and further work

Tweety is ...

- ▶ ... a multi-purpose framework for Knowledge Representation
- ▶ ... Open Source and licensed under GNU GPL v3.0
- ▶ ... being constantly improved

Current Work:

- ▶ Plugin architecture
- ▶ Command line interface
- ▶ Description logic library

Future Work:

- ▶ Web interface
- ▶ More KR formalisms



Thank you for your attention

Join and participate: <http://tweetyproject.org>