Implementing KR Approaches with Tweety

Matthias Thimm

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

KR 2018

Motivation 1/3

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

Motivation 2/3

- Even theoretical work can benefit from implementations
  - allows other people to easily do examples
  - proof-of-concept to show feasibility
  - compare behaviour with other approaches, not just with toy examples

- KR2016: 19 of 74 (long and short) papers mentioned implementations ($\approx$ 26%, compare to e. g. Semantic Web conferences with $>$ 90%)

Motivation 3/3

- 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 37 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*, *BeliefBaseReasoner*,. . .
  - Tools for dealing with sets, graphs, mathematical expressions, mathematical optimization,. . .
  - Command line interface
- Basic implementations of over 23 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 Oct 15, 2018)

- 943 SVN commits
- First SVN commit: 3 July 2010
- Current Version: 1.11
- 18 developers in total
- 2–3 active developers at any time

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

- Total Physical Source Lines of Code: 71,563
- Development Effort Estimate, Person-Years: 17.72
- Total Estimated Cost to Develop: $ 2,393,672

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

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

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

Overview

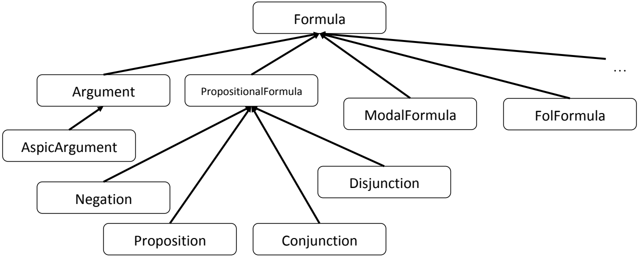
Knowledge Representation concepts are mapped 1:1 to Java classes/interfaces:

- `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

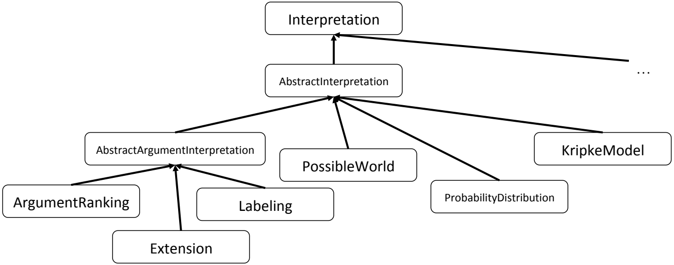


Example

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

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

Interpretation

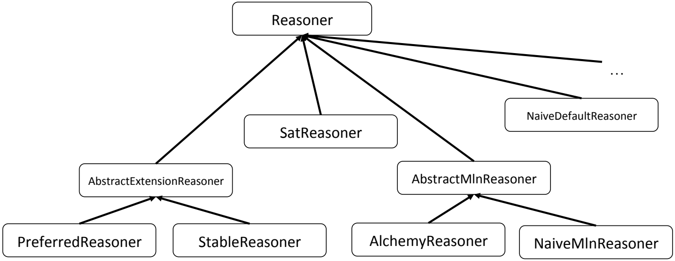


Example

```
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.getArgumentsOfStatus(ArgumentStatus.IN));
```

BeliefBaseReasoner



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);
SatSolver.setDefaultSolver(new Sat4jSolver());
BeliefBaseReasoner<PlBeliefSet> r = new SatReasoner();
System.out.println( r.query(bs, 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

More Code Examples: Mathematical Optimization

```
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:

$\min x^2 + y^2$

s.t. $x + y = 1$

Outline

1. Introduction

2. Installation and Usage
   - Installation
   - Package overview

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

Outline

1. Introduction

2. Installation and Usage
   - Installation
   - Package overview

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 (or higher) 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
```

Installation in 30 seconds 2/4

3. Add Tweety as dependency

```
<project ...>
  ...
  <dependencies>
    ...
    <dependency>
     <groupId>net.sf.tweety</groupId>
     <artifactId>tweety-full</artifactId>
     <version>1.10</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/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

1. Introduction

2. Installation and Usage
   - Installation
   - Package overview

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

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`)
- 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`)
- 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`)
- 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`)

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

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

Basic classes 1/3

- 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. `BeliefBaseReasoner`: 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. . . .

Basic classes 2/3

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
    <B extends BeliefBase, S extends Formula> {

    public boolean satisfies(S formula);
    public boolean satisfies(Collection<S> formulas);
    public boolean satisfies(B beliefBase);
}
```

Basic classes 3/3

```
public interface Reasoner
    <O,B extends BeliefBase,F extends Formula> {

    public O query(B beliefbase, F formula);
}

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 1/2

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);
}
```

Parser and Writer 2/2

```java
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) { ... }
}
```

Outline

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

5. Basics: Mathematics and Graphs

6. Advanced topics: Computational Argumentation

7. Advanced topics: Inconsistency Measurement

8. Summary and Conclusion

Propositional Logic - Recap 1/4

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* $\mathcal{L}(\mathrm{At})$ for At is the minimal set $\mathcal{L}$ satisfying

1. At $\subseteq \mathcal{L}$,
2. $\top, \bot \in \mathcal{L}$ (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

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

Let 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 $\mathrm{At} = \{a, b, c\}$. The interpretation I_1 of At given by

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

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

Propositional Logic - Recap 3/4

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

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

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

Furthermore, for every interpretation I it holds that $I \models^{\mathsf{P}} \top$ and $I \not\models^{\mathsf{P}} \bot$.

Propositional Logic - Recap 4/4

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

Outline

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
5. Basics: Mathematics and Graphs
6. Advanced topics: Computational Argumentation
7. Advanced topics: Inconsistency Measurement
8. Summary and Conclusion

Propositional Logic in Tweety

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(Conjuctable 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);
```

Parse formulas

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;   } ... }
```

Outline

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
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 $\Phi \models^{\mathsf{P}} \alpha$ 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));
```

Outline

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
6. Advanced topics: Computational Argumentation
7. Advanced topics: Inconsistency Measurement
8. Summary and Conclusion

Mathematics and Graphs

- 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

Outline

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

6. Advanced topics: Computational Argumentation

7. Advanced topics: Inconsistency Measurement

8. Summary and Conclusion

Working with mathematical terms

- 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 continous
  - 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

Outline

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
6. Advanced topics: Computational Argumentation
7. Advanced topics: Inconsistency Measurement
8. Summary and Conclusion

KR and optimization problems

- 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

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

(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);
```

Outline

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

6. Advanced topics: Computational Argumentation

7. Advanced topics: Inconsistency Measurement

8. Summary and Conclusion

Graphs in Tweety

- The package `net.sf.tweety.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.)

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
   - Introduction to Computational Argumentation
   - Computational Argumentation in Tweety

7. Advanced topics: Inconsistency Measurement

8. Summary and Conclusion

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
   - Introduction to Computational Argumentation
   - Computational Argumentation in Tweety
7. Advanced topics: Inconsistency Measurement
8. Summary and Conclusion

Abstract Argumentation

Definition (Abstract Argumentation Framework)

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

An extension E is a set $E \subseteq \mathsf{Arg}$ 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

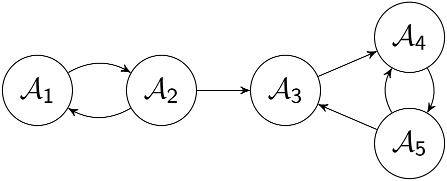
3. every argument $\mathcal{C}$ that is *defended* by E, belongs to E

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.

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
   - Introduction to Computational Argumentation
   - Computational Argumentation in Tweety
7. Advanced topics: Inconsistency Measurement
8. Summary and Conclusion

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`)
- Deductive Argumentation (`net.sf.tweety.arg.deductive`)
- Social Abstract Argumentation (`net.sf.tweety.arg.social`)
- Structured Argumentation Frameworks (`net.sf.tweety.arg.saf`)
- Simple Logic Argumentation (`net.sf.tweety.arg.simplelogicdeductive`)
- 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`)

Abstract Argumentation 1/2

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)
- Also available: general bridge to include any `probo`-compatible solver (the interface standard of the argumentation competition)

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
   - Introduction to Inconsistency Measurement
   - Inconsistency Measures in Tweety

8. Summary and Conclusion

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
    - Introduction to Inconsistency Measurement
    - Inconsistency Measures in Tweety
8. Summary and Conclusion

Inconsistency Measurement 1/3

Compare the following propositional knowledge bases

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

- 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 $\mathbb{K}(\mathsf{At})$ be the set of all knowledge bases of signature At.

Definition

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

Inconsistency Measurement 2/3

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 $\mathsf{MI}(\mathcal{K})$ 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 $M \in \mathsf{MI}(\mathcal{K})$ with $\alpha \in M$. Let $\mathsf{Free}(\mathcal{K})$ denote the set of all free formulas of $\mathcal{K}$.

Inconsistency Measurement 3/3

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 $\alpha \in \mathsf{Free}(\mathcal{K})$ we have $\mathcal{I}(\mathcal{K}) = \mathcal{I}(\mathcal{K} \setminus \{\alpha\})$.

Example

Define $\mathcal{I}_{\mathsf{MI}}(\mathcal{K}) = |\mathsf{MI}(\mathcal{K})|$
Then we have

$$\mathcal{I}_{\mathsf{MI}}(\{a, \neg a, c\}) = 1$$
$$\mathcal{I}_{\mathsf{MI}}(\{a, b, \neg a \wedge \neg b\}) = 2$$

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
   - Introduction to Inconsistency Measurement
   - Inconsistency Measures in Tweety
8. Summary and Conclusion

Inconsistency Measures in Tweety 1/2

General interface:

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

Available implementations: $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_d$, $\mathcal{I}_{\mathsf{MI}^\mathsf{C}}$, $\mathcal{I}_\eta$, $\mathcal{I}_c$, $\mathcal{I}_{mc}$, $\mathcal{I}_p$, $\mathcal{I}_{hs}$, $\mathcal{I}^{\Sigma}_{\mathsf{dalal}}$, $\mathcal{I}^{\mathsf{max}}_{\mathsf{dalal}}$, $\mathcal{I}^{\mathsf{hit}}_{\mathsf{dalal}}$, $\mathcal{I}_{D_f}$, $\mathcal{I}_{P_m}$, $\mathcal{I}_{mv}$, $\mathcal{I}_{nc}$, $\ldots$

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

Inconsistency Measures in Tweety 2/2

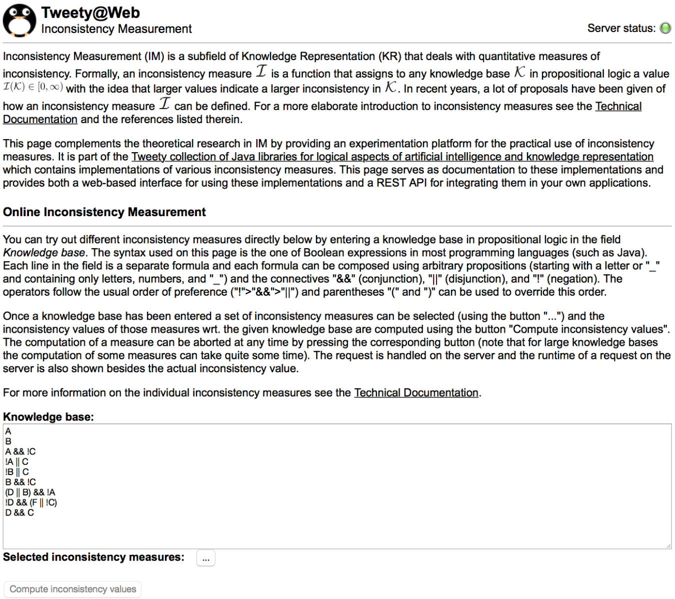
Usage:

```
PlMusEnumerator.setDefaultEnumerator(
    new NaiveMusEnumerator<PropositionalFormula>
        (new Sat4jSolver()));

MiInconsistencyMeasure<PropositionalFormula> miInc =
    new MiInconsistencyMeasure<PropositionalFormula>
        (PlMusEnumerator.getDefaultEnumerator());

PlBeliefSet kb = ...
System.out.println(miInc.inconsistencyMeasure(kb));
```

Web interface: `http://tweetyproject.org/w/incmes`



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.

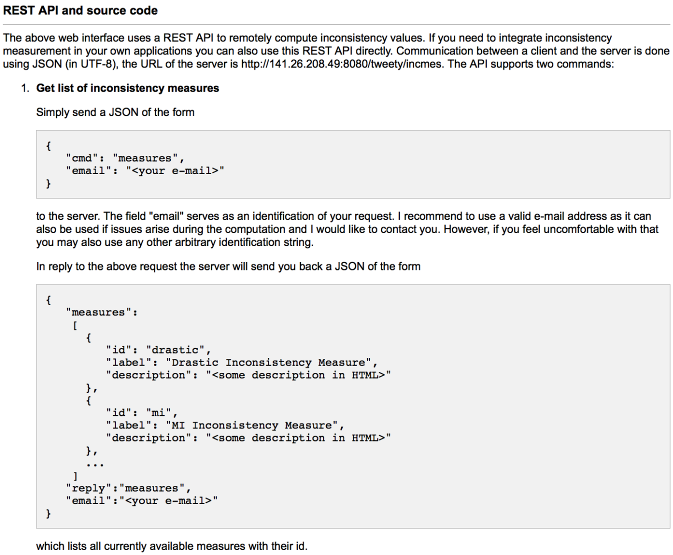
For more information on the individual inconsistency measures see the Technical Documentation.

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



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.

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

KR/AI 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 analytically, but experimental evaluation helps for trial-and-error purposes

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

Final Remarks and further work

Tweety is . . .

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

Current Work:

- Description Logic library
- Abstract Dialectical Frameworks library
- Machine Learning library

Future Work:

- Web interface
- More KR formalisms

Thank you for your attention

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