Tweety: A Comprehensive Collection of Java Libraries for Logical Aspects of Artificial Intelligence and Knowledge Representation

Matthias Thimm
Institute for Web Science and Technologies (WeST)
University of Koblenz-Landau
Germany

Abstract

This paper presents Tweety, an open source project for scientific experimentation on logical aspects of artificial intelligence and particularly knowledge representation. Tweety provides a general framework for implementing and testing knowledge representation formalisms in a way that is familiar to researchers used to logical formalizations. This framework is very general, widely applicable, and can be used to implement a variety of knowledge representation formalisms from classical logics, over logic programming and computational models for argumentation, to probabilistic modeling approaches. Tweety already contains over 15 different knowledge representation formalisms and allows easy computation of examples, comparison of algorithms and approaches, and benchmark tests. This paper gives an overview on the technical architecture of Tweety and a description of its different libraries. We also provide two case studies that show how Tweety can be used for empirical evaluation of different problems in artificial intelligence.

1 Introduction

Knowledge Representation and Reasoning (KR) [Brachman and Levesque, 2004] is an important subfield in Artificial Intelligence (AI) that deals with issues regarding formalizing knowledge in such a way that machines can read, understand, and reason with it. Nowadays, KR has a lot of applications within, e. g., the semantic web [Antoniou and van Harmelen, 2004], as a lot of work on description logics [Baader et al., 2003] and ontologies originate from this field (at least the technical or computer-science-oriented perspectives on those). Apart from that, more fundamental work in KR deals with issues regarding uncertainty of beliefs, dynamics of belief, and defeasible reasoning. Many branches of research in knowledge representation and reasoning is theoretical in nature and researchers usually do not put effort in implementation and empirical evaluation. To address this issue we present in this field study the Tweety libraries for logical aspects of artificial intelligence and knowledge representation.

Approaches to knowledge representation follow almost always a specific pattern. Starting from a formal syntax one can build formulas which are collected in knowledge bases. Using knowledge bases one can derive new information using either the underlying semantics of the language or a specific reasoner. For example, propositional logic is the most basic form for knowledge representation. Given some set of propositions (or atoms) one can build complex formulas using disjunction, conjunction, or negation. A set of propositional formulas, i. e., a knowledge base, can be used to derive new propositional formulas as conclusions. For instance, this can be done using the standard model-theoretic semantics of propositional logic or more sophisticated reasoning techniques such as paraconsistent reasoning. Most logical approaches to knowledge representation such as first-order logic, description logics, defeasible logics, default logics, probabilistic logics, fuzzy logics, etc. follow this pattern. Moreover, many other formalisms which are not so obviously rooted in logic such as abstract argumentation or Bayes nets can also be cast into this framework. For example, for abstract argumentation frameworks [Dung, 1995], a knowledge base is given by a conjunction of attack statements between arguments and different kinds of semantics such as grounded or stable semantics determine how sets of arguments can be derived from a knowledge base.

The Tweety libraries support the implementation of such approaches by providing a couple of abstract classes and interfaces for components such as `Formula`, `BeliefBase`, and `Reasoner`. Furthermore, many strictly logic-based approaches to knowledge representation can also utilize further classes such as `Predicate`, `Atom`, and `Variable`, to name just a few. Currently, Tweety already contains implementations of over 15 different approaches to knowledge representation such as propositional logic, first-order logic, several approaches to probabilistic logics, and several approaches to computational models of argumentation.

In this paper, besides giving an overview on the technical details of Tweety and its libraries, we also report on two case studies that use Tweety as a framework for experimentation and empirical evaluation. The first study is on inconsistency measurement for probabilistic logics [Thimm, 2011; 2013b]. In general, probabilistic logics are concerned with using quantitative uncertainty for non-monotonic reasoning. Naturally, these approaches are computationally hard and not easy to understand, as the underlying reasoning mechanisms are quite complicated. Consequently, implementations serve well to understand examples and to (in-)validate conjectures. Our second case study is about strategic argumentation in multi-agent systems [Thimm and Garcia, 2010; Rienstra, Thimm, and Oren, 2013]. Similarly, when defining agent models and negotiation strategies in such an environment, effects that occur on a larger scale are hard to predict by hand. Moreover, just an analytical evaluation of different negotiation strategies is often also simply too weak to provide meaningful insights [Rienstra, Thimm, and Oren, 2013]. For that reason Tweety can also be used as a tool for empirical evaluation as it has been done in [Rienstra, Thimm, and Oren, 2013] to provide average performance results on a series of experiment runs in random settings. Further works that use Tweety for implementing knowledge representation formalisms or for empirical evaluation are, e. g., [Thimm and Kern-Isberner, 2008; Thimm and Garcia, 2010; Krümpelmann et al., 2011; Thimm, 2011; Kern-Isberner and Thimm, 2012; Thimm, 2012; 2013b; Rienstra, Thimm, and Oren, 2013; Thimm, 2013a; Krümpelmann and Kern-Isberner, 2012].

The rest of this paper is organized as follows. Section 2 gives an overview on the architecture of Tweety and Section 3 presents some technical details on its different libraries. In Section 4 two case studies are presented that show how Tweety can be used for evaluation in scientific research. Section 5 concludes with a summary and pointers to future work.

2 Technical Overview

Tweety is organized as a modular collection of Java libraries with a clear dependence structure. The programming language Java has been chosen as it is easy to understand, commonly used, and platform-independent. Each knowledge representation formalism has a dedicated Tweety library (ranging from a library on propositional logic to libraries on computational models of argumentation) which provides implementations for both syntactic and semantic constructs of the given formalism as well as reasoning capabilities. Several libraries provide basic functionalities that can be used in other projects. Among those is the *Tweety Core* library which contains abstract classes and interfaces for all kinds of knowledge representation formalisms. Furthermore, the library *Math* contains classes for dealing with mathematical sub-problems that often occur, in particular, in probabilistic approaches to reasoning. Most other Tweety projects deal with specific approaches to knowledge representation. In the next section, we have a closer look on the individual libraries.

Each Tweety library is organized as a Maven[1] project (Maven is a tool for organizing dependencies between projects, building, and deploying). Most libraries can be used right away as they only have dependencies to other Tweety libraries. Some libraries provide bridges to third-party libraries such as numerical optimization solvers which are not automatically found by Maven and have to be installed beforehand. However, all necessary third-party libraries can be installed by executing a single install file located within the Tweety distribution.

In order to use and develop with Tweety we recommend using the Eclipse IDE[2] and its Maven plugin[3]. As all Tweety libraries are organized as Maven projects they can all be easily imported and used for other projects within Eclipse. Furthermore, pre-compiled JARs for every library can be downloaded from the Tweety homepage[4] and directly used in other projects. A third way of using the functionalities of Tweety is by using its *Command Line Interface* which is currently under development.

Currently, Tweety contains 161 Java packages which themselves contain 794 Java classes in overall 87421 lines of code. The average cyclomatic complexity number per method (CCN) [McCabe, 1976] is 2.87. This means, that every method roughly contains two to three if-else statements thus reducing complexity on a method-basis and emphasizing the modular nature of Tweety. Furthermore, the average code-to-comment ratio in Tweety is 2.13, meaning that for roughly every two lines of code one line of comment is given.

3 Libraries

In the following we give a detailed description of the currently available libraries within Tweety. An overview of these libraries is given in Table 1 which provides both the name of a library and its Java root package name. Furthermore, the final column lists references to original literature and the implemented reasoning mechanisms and solvers. There, a dagger (†) indicates that a the particular reasoning mechanism has been directly implemented from the original literature, a double dagger (‡) means that a wrapper for the existing original implementation is provided, and an asterisk (*) refers to related literature.

General Libraries

The *General* libraries of Tweety provide basic functionalities and utility classes for all other Tweety classes.

Tweety Core The *Tweety Core* library contains abstract classes and interfaces for various knowledge representation concepts. Among the most important ones are

`Formula` A formula of a representation formalism.

`BeliefBase` Some structure containing beliefs.

`BeliefSet` A set of beliefs, i. e., a set of formulas, it is the most commonly used class derived from `BeliefBase`. Please note that we follow the Java guideline for naming a class containing a set of beliefs a *belief set* (it contains a finite unordered set of elements), opposed to the naming convention in belief dynamics where a *belief set* is usually deductively closed. In terms of belief dynamics research the class `BeliefSet` actually represents a *belief base*.

`Signature` The signature of a representation formalism.

[1] http://maven.apache.org

[2] http://www.eclipse.org

[3] http://maven.apache.org/eclipse-plugin.html

[4] http://www.mthimm.de/projects/tweety/

Library	Project root package	Notes
General Libraries		
Tweety Core	`net.sf.tweety`	
Command Line Interface	`n.s.t.cli`	
Plugin	`n.s.t.plugin`	*JSPF[a]
Math	`n.s.t.math`	‡lpsolve[b], ‡OpenOpt[c], ‡Apache simplex[d], ‡Choco[e], †Gradient descent
Graphs	`n.s.t.graphs`	
Logic Libraries		
Logic Commons	`n.s.t.logics.commons`	†[Grant and Hunter, 2006], †[Hunter and Konieczny, 2006]
Propositional Logic	`n.s.t.logics.pl`	‡Sat4j[f]
First-Order Logic	`n.s.t.logics.fol`	*[Brachman and Levesque, 2004]
Conditional Logic	`n.s.t.logics.cl`	†[Pearl, 1990], †[Kern-Isberner, 2001]
Relational Conditional Logic	`n.s.t.logics.rcl`	†[Kern-Isberner and Thimm, 2012]
Probabilistic Conditional Logic	`n.s.t.logics.pcl`	†[Rödder, 2000], †[Thimm, 2013b], †[Potyka, 2014]
Relational Prob. Conditional Logic	`n.s.t.logics.rpcl`	†[Kern-Isberner and Thimm, 2010]
Markov Logic	`n.s.t.logics.ml`	†[Richardson and Domingos, 2006] ‡Alchemy[g]
Epistemic Logic	`n.s.t.logics.el`	*[Fagin et al., 1995]
Description Logic	`n.s.t.logics.dl`	*[Baader et al., 2003]
Logic Translators	`n.s.t.logics.translators`	
Logic Programming Libraries		
Answer Set Programming	`n.s.t.lp.asp`	‡Clingo[h], ‡DLV[i]
Dynamics in Answer Set Programming	`n.s.t.lp.asp.beliefdynamics`	†[Krümpelmann and Kern-Isberner, 2012]
Nested Logic Programming	`n.s.t.lp.nlp`	*[Lifschitz, Tang, and Turner, 1999]
Argumentation Libraries		
Abstract Argumentation	`n.s.t.arg.dung`	†[Dung, 1995], †[Baroni, Caminada, and Giacomin, 2011]
Deductive Argumentation	`n.s.t.arg.deductive`	†[Besnard and Hunter, 2001], †[Besnard and Hunter, 2006]
Structured Argumentation Frameworks	`n.s.t.arg.saf`	†[Thimm and Garcia, 2010]
Defeasible Logic Programming	`n.s.t.arg.delp`	†[Garcia and Simari, 2004], †[Stolzenburg et al., 2003]
Logic Programming Argumentation	`n.s.t.arg.lp`	†[Schweimeier and Schroeder, 2003]
Probabilistic Argumentation	`n.s.t.arg.prob`	†[Thimm, 2012]
Agent Libraries		
Agents	`n.s.t.agents`	*[Weiss, 2013]
Dialogues	`n.s.t.agents.dialogues`	†[Thimm and Garcia, 2010], †[Rienstra, Thimm, and Oren, 2013]
Other Libraries		
Action and Change	`n.s.t.action`	†[Gelfond and Lifschitz, 1998]
Belief Dynamics	`n.s.t.beliefdynamics`	†[Hansson, 2001], †[Fermé and Hansson, 1999], †[Krümpelmann et al., 2011]
Machine Learning	`n.s.t.machinelearning`	‡LIBSVM[j]
Preferences	`n.s.t.preferences`	*[Walsh, 2007], †[Thimm, 2013a]

Table 1: Overview on the Tweety libraries (the prefix `n.s.t` stands for `net.sf.tweety`)

[a] https://code.google.com/p/jspf/
[b] http://lpsolve.sourceforge.net
[c] http://openopt.org
[d] http://commons.apache.org/math
[e] http://www.emn.fr/z-info/choco-solver/
[f] http://www.sat4j.org
[g] http://alchemy.cs.washington.edu
[h] http://potassco.sourceforge.net
[i] http://www.dlvsystem.com
[j] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Interpretation An interpretation that evaluates the truth of formulas.

Reasoner Implements a specific reasoning strategy to answer queries for a representation formalism.

Parser, Writer For reading/writing formulas and belief sets.

Most other Tweety libraries provide specific implementations of the above abstract classes and interfaces for their specific representation formalisms. For example, the library *Propositional Logic* implements `Formula` by `PropositionalFormula` (which is recursively defined using conjunction, disjunction, and negation) and `Interpretation` by `PossibleWorld`. In this way, the classical approach to formally define a logical language via syntax and semantics has a one-to-one correspondence with its implementation in Tweety.

Besides the above mentioned abstract classes and interfaces, *Tweety Core* provides abstract implementations of several other knowledge representation concepts and several utility classes for working with sets, subsets, vectors, and general rules.

Plugin The *Plugin* library provides classes for implementing Tweety plugins that can be used by, e. g., the *Command Line Interface*. This library makes use of the *Java Simple Plugin Framework (JSPF)*[5]. Using these classes one can encapsulate the functionalities of a specific knowledge representation formalism and expose them in the same way to user interfaces. The most important class is the abstract class `AbstractTweetyPlugin` which is the basis for developing plugins. Please note that the *Plugin* library is currently in an experimental phase.

Command Line Interface All Tweety libraries can be accessed programmatically in Java through their API (*Application Programming Interface*). However, for non-programmers this way of utilizing the libraries is not very convenient. Using the *Plugin* library the *Command Line Interface* library provides a general command line interface for many Tweety libraries. Every library can expose its functionality through a Tweety plugin that can be plugged into the command line interface and accessed in a uniform way. Please note that the *Command Line Interface* library is currently in an experimental phase.

Math Many algorithms for knowledge representation and reasoning are based on mathematical methods such as optimization techniques. The *Math* library encapsulates those mathematical methods and exposes them through simple interfaces to other libraries for realizing these algorithms. At the core, the *Math* library contains classes for representing mathematical terms (such as `Constant`, `Variable`, `Product`, `Logarithm`) and statements (such as `Equation`). Using these constructs one can represent, e. g., constraint satisfaction problems (`ConstraintSatisfactionProblem`) and optimization problems (`OptimizationProblem`). Through the `Solver` interface the *Math* library provides bridges to several third-party solvers such as the *ApacheCommons* Simplex-algorithm[6], the *OpenOpt* solvers[7], or Choco[8].

[5]https://code.google.com/p/jspf/

Graphs The *Graphs* library contains a simple graph implementation with utility functions as it can be used, e. g., to represent abstract argumentation frameworks (see *Abstract Argumentation* library).

Logic Libraries

The *Tweety Logic* libraries (located under the package `net.sf.tweety.logics`) provide implementations for various knowledge representation formalisms based on classical logics (propositional logic and first-order logic) and non-classical logics such as conditional logic, probabilistic logics, epistemic logics, or description logic. Each library follows a strict approach in defining the formalism by implementing the abstract classes and interfaces `Formula`, `BeliefBase`, `Interpretation`,... from the *Tweety Core* library. Each library contains a sub-package `syntax` which contains the elements to construct formulas of the formalism and a sub-package `semantics` which contains elements for realizing the semantics of the formalism. Besides these two common sub-packages many libraries also contain parsers for reading formulas from file and reasoner that implement a specific reasoning approach.

Logic Commons The *Logic Commons* library contains abstract classes and interfaces which further refine the general `Formula` interface from the *Tweety Core* library. Among these refinements are several concepts that are shared among a great number of knowledge representation formalism such as `Predicate`, `Variable` or `Atom`.

Propositional Logic The *Propositional Logic* library provides an implementation of classical propositional logic. Propositional formulas can be constructed using, e. g., classes `Conjunction` or `Disjunction` and propositional formulas can be put into a knowledge base of type `PlBeliefSet`. Currently, the *Propositional Logic* library supports two different reasoners. The first is a simple brute force approach that directly follows the definition of classical entailment, i. e., in order to prove a given propositional formula wrt. a given set of propositional formulas all possible worlds are enumerated and tested. Obviously, this reasoner only works for small examples but is useful when one is interested in all models of a knowledge base. The second supported reasoner incorporates the Sat4j reasoner[9]. Other SAT-solvers can be added in a straightforward way.

First-Order Logic This library contains an implementation of first-order logic as a knowledge representation formalism. Both the *Propositional Logic* library and the *First-Order Logic* library are used by many other libraries of knowledge representation formalisms.

[6]http://commons.apache.org/math

[7]http://openopt.org

[8]http://www.emn.fr/z-info/choco-solver/

[9]http://www.sat4j.org

Conditional Logic The *Conditional Logic* library extends the *Propositional Logic* library by *conditionals*, i. e., non-classical rules of the form $(B \mid A)$ ("A usually implies B"), cf. [Nute and Cross, 2002]. In the literature, several different semantics and reasoning approaches for conditional logics have been proposed and this library can be used to easily compare their reasoning behavior. Currently, the *Conditional Logic* library implements interpretations in the form of ranking functions [Spohn, 1988] and conditional structures [Kern-Isberner, 2001], and provides reasoner based on z-ranking [Goldszmidt and Pearl, 1996] and c-representations [Kern-Isberner, 2001].

Relational Conditional Logic Similar to the *Conditional Logic* library the *Relational Conditional Logic* extends the *First-Order Logic* libraries with relational conditionals (i. e. conditionals that may contain first-order formulas), cf. [Delgrande, 1998; Kern-Isberner and Thimm, 2012]. Currently, this library contains an implementation of the relational c-representation reasoning approach of [Kern-Isberner and Thimm, 2012].

Probabilistic Conditional Logic This library further extends the *Conditional Logic* library by extending conditionals to *probabilistic conditionals* of the form $(B \mid A)[p]$ ("A usually implies B with probability p"), cf. [Rödder, 2000]. Besides a naive implementation of probabilistic reasoning based on the principle of maximum entropy [Paris, 1994] this library also contains several classes for analyzing and repairing inconsistent sets of probabilistic conditionals, cf. [Thimm, 2011; 2013b]. We will discuss this package in more detail in Section 4.

Relational Probabilistic Conditional Logic By combining both the *Relational Conditional Logic* and *Probabilistic Conditional Logic* libraries the *Relational Probabilistic Conditional Logic* library introduces relational conditionals with probabilities, cf. [Kern-Isberner and Thimm, 2010]. It implements both the *averaging* and *aggregating* semantics from [Kern-Isberner and Thimm, 2010] and also allows for lifted inference as proposed in [Thimm, 2011].

Markov Logic This library builds on the *First-Order Logic* library to implement *Markov Logic*, an extension of first-order logic with weights to allow for probabilistic reasoning, cf. [Richardson and Domingos, 2006]. It provides several propriety sampling-based reasoner and a bridge to the Alchemy reasoner[10].

Epistemic Logic This library extends the *Propositional Logic* library with modal operators for epistemic logic and its semantics with accessibility relations and Kripke models. Please note that the *Epistemic Logic* library is currently in an experimental phase.

Description Logic The *Description Logic* library provides a general description logic implementation [Baader et al., 2003] based on the *First-Order Logic* library. Please note that the *Description Logic* library is currently in an experimental phase.

[10]http://alchemy.cs.washington.edu

Logic Translators This library provides the abstract class `Translator` that provides basic functionalities to implement translators between different knowledge representation formalisms. Currently, the *Logic Translators* library contains translators between first-order logic and answer set programming, between nested logic programming and answer set programming, and between propositional logic and first-order logic.

Logic Programming Libraries

The *Logic Programming* libraries (located under the package `net.sf.tweety.lp`) provide implementations of knowledge representation formalisms based on logic programming.

Answer Set Programming The *Answer Set Programming* library provides classes for representing extended logic programs [Gelfond and Leone, 2002]. Answer set programs are logic programs of the form $A \leftarrow B_1, \ldots, B_m$ with first-order literals $A, B_1, \ldots, B_m$ where the body literals $B_1, \ldots, B_m$ may also have a default negation not. This library provides bridges to several established solvers such as DLV[11], DLV Complex[12], and Clingo[13].

Dynamics in Answer Set Programming This library extends the *Answer Set Programming* library by introducing revision and update approaches. The library contains implementations of the approaches introduced in [Krümpelmann and Kern-Isberner, 2012; Delgrande, Schaub, and Tompits, 2007] and also revision approaches based on argumentation.

Nested Logic Programming This library contains an implementation of nested logic programs which allow for complex first-order formulas to appear in logic programming rules [Lifschitz, Tang, and Turner, 1999].

Argumentation Libraries

The argumentation libraries (located under the package `net.sf.tweety.arg`) are one of the most mature libraries of Tweety and contain a wide variety of implementations of different approaches to computational argumentation.

Abstract Argumentation This library implements abstract argumentation as proposed in [Dung, 1995]. An abstract argumentation framework is a directed graph (A, Att) where A is interpreted as a set of arguments and an edge $(A, A') \in Att$ is an attack of A on A'. The library provides implementations of the mostly used semantics and their corresponding reasoner, both in terms of extensions (an extension is a set of arguments that is regarded as accepted by a semantics) and labelings (a labeling is a function with a three-valued truth assignment to each argument). Several utility classes for generating random argumentation frameworks complement this library.

[11]http://www.dlvsystem.com

[12]https://www.mat.unical.it/dlv-complex

[13]http://potassco.sourceforge.net

Deductive Argumentation The *Deductive Argumentation* library provides an implementation of the approach proposed in [Besnard and Hunter, 2001]. In deductive argumentation, an argument is composed of a set of propositional formulas that derive the claim of the argument. Attack between arguments is derived from classical unsatisfiability.

Structured Argumentation Frameworks This library implements the approach of structured argumentation frameworks as proposed in [Thimm and Garcia, 2010]. In structured argumentation frameworks arguments are composed of subarguments and a conclusion.

Defeasible Logic Programming This library provides an implementation of Defeasible Logic Programming (DeLP) [Garcia and Simari, 2004]. In DeLP knowledge bases contain strict and defeasible rules and facts, similar to knowledge representation formalisms for logic programming. Defeasible rules can be collected in arguments and compared by generalized specificity [Stolzenburg et al., 2003].

Logic Programming Argumentation This library provides an implementation of the argumentation approach of [Schweimeier and Schroeder, 2003] which is also based on logic programming techniques.

Probabilistic Argumentation The *Probabilistic Argumentation* library extends the *Abstract Argumentation* library with non-classical semantics based on probabilistic assessments [Thimm, 2012].

Agent Libraries

The agent libraries (located under the package `net.sf.tweety.agents`) provide a framework for analyzing and simulating interactions between agents.

Agents This general library contains an abstract formalization of agents and multi-agent systems. Classes such as `Agent`, `Environment`, `MultiAgentSystem`, and `Protocol` can be used to set up and simulate a system of agents within an environment. This library has a specific focus on the simulation aspect and provides classes such as `MultiAgentSystemGenerator` and `GameSimulator` that allow the automatic generation of test scenarios and their evaluation.

Dialogues The library *Dialogues* extends the *Agents* library with the capability of simulating dialogues between agents, as they are investigated in the context of argumentation in multi-agent systems [Karunatillake et al., 2009]. It also provides an implementation of agents with an opponent model as proposed in [Rienstra, Thimm, and Oren, 2013]. We will discuss this package in more detail in Section 4.

Other Libraries

The above discussed libraries constitute the core of Tweety by providing implementations of several knowledge representation formalisms. This collection is complemented by some further libraries that relate either to topics that do not strictly belong to the field of knowledge representation (such as the *Machine Learning* library) or can be applied across several different knowledge representation formalisms (such as the *Belief Dynamics* library).

Action and Change The *Action and Change* library implements several action languages and their dynamics from [Gelfond and Lifschitz, 1998].

Belief Dynamics This library provides a general implementation for various approaches to belief (base) revision and update [Hansson, 2001]. It provides interfaces and several implementations of many concepts used in belief dynamics such as `BaseRevisionOperator`, `BaseContractionOperator`, `IncisionFunction`, and `LeviBaseRevisionOperator`. Those classes are defined in such a general way that they can be used not only to implement belief dynamics for propositional logic but also for other knowledge representation formalisms implementing the corresponding Tweety interfaces. This library contains also specific revision approaches such as selective revision [Fermé and Hansson, 1999] and argumentative selective revision [Krümpelmann et al., 2011].

Machine Learning The *Machine Learning* library provides several abstract concepts that can be used in a machine learning context such as `Observation`, `Classifier`, and `CrossValidator`. It contains also an implementation of support vector machines utilizing LIBSVM[14].

Preferences This library contains classes for representing preference orders and approaches for aggregating them [Walsh, 2007]. It also contains an implementation of the dynamic preference aggregation approach proposed in [Thimm, 2013a].

4 Case Studies and Evaluation

In this section we discuss two case studies that make use of Tweety as a platform for experimentation and empirical evaluation. The first case study is on inconsistency handling for probabilistic logics [Thimm, 2011; 2013b] while the second study is on strategic argumentation in multi-agent systems [Thimm and Garcia, 2010; Rienstra, Thimm, and Oren, 2013].

Inconsistency Handling for Probabilistic Logics

In order to motivate the work described in this section we give a brief introduction into probabilistic conditional logic and its inconsistency measures, cf. [Thimm, 2011; 2013b].

For propositional formulas ϕ, ψ and a real-value $p \in [0, 1]$ we call $(\phi \mid \psi)[p]$ a *probabilistic conditional*. A probabilistic conditional $(\phi \mid \psi)[p]$ represents a specific form of defeasible rule and has the intuitive meaning "if ψ is true then ϕ is true with probability p". A (probabilistic conditional) *knowledge base* $\mathcal{K}$ is a set of probabilistic conditionals. Semantics are given to probabilistic conditionals by probability functions $P : \Omega \to [0, 1]$ with Ω being the set of interpretations (possible worlds) of the underlying propositional logic (We assume that the set of propositions is finite and so is the

[14]http://www.csie.ntu.edu.tw/~cjlin/libsvm/

set of possible worlds). A probability function P *satisfies* a conditional $(\phi \,|\, \psi)[p]$ if and only if $P(\phi \wedge \psi) = pP(\psi)$ (the probability of a formula is defined to be the sum of the probabilities of all possible worlds satisfying it). Note that this follows the definition of conditional probability ($P(\phi \mid \psi) = P(\phi \wedge \phi)/P(\psi) = p$) as long as $P(\psi) \neq 0$. In order to avoid a case differentiation for $P(\psi) = 0$ we use the above definition, cf. [Paris, 1994]. A probability function P satisfies a knowledge base $\mathcal{K}$ if and only if it satisfies all its probabilistic conditionals. A knowledge base is consistent if such a probability function exists.

Example 1 *Consider* $\mathcal{K} = \{(f \,|\, b)[0.9], (b \,|\, p)[1], (f \,|\, p)[0.01]\}$ *with the intuitive meaning that birds (b) usually (with probability* 0.9*) fly (f), that penguins (p) are always birds, and that penguins usually do not fly (only with probability* 0.01*). The knowledge base* $\mathcal{K}$ *is consistent as a probability function satisfying it can easily be constructed, cf. [Thimm, 2013b]. Note that, e. g., the knowledge base* $\mathcal{K} = \{(x \,|\, y)[0.9], (y \,|\, \top)[0.9], (x \,|\, \top)[0.2]\}$ *is inconsistent (*$\top$ *is a logical tautology): considering just the conditionals* $(x \,|\, y)[0.9]$ *and* $(y \,|\, \top)[0.9]$ *we obtain that* x *has to be at the least probability* 0.81 *which is inconsistent with stating that* x *has probability* 0.2.

In order to deal with inconsistent knowledge bases the work [Thimm, 2013b] proposes inconsistency measures as a tool for analyzing inconsistencies. An inconsistency measure is a function $\mathcal{I}$ that takes a knowledge base $\mathcal{K}$ and computes an inconsistency value $\mathcal{I}(\mathcal{K}) \in [0, \infty)$ with the intuitive meaning that a larger value indicates a more severe inconsistency (and $\mathcal{I}(\mathcal{K}) = 0$ means that $\mathcal{K}$ is consistent). See [Thimm, 2013b] for more details, some rationality postulates on inconsistency measurement, and specific approaches.

The inconsistency measurement framework for probabilistic logics has been implemented in the *Probabilistic Conditional Logic* library of Tweety (sub-package `net.sf.tweety.logics.pcl.analysis`). However, as inconsistency measurement is a broader topic that can also be used in other knowledge representation formalisms such as classical logics [Grant and Hunter, 2006], the concept *inconsistency measure* is already implemented in the *Logic Commons* library (sub-package `net.sf.tweety.logics.commons.analysis`) as a very general interface (only a simplified version is shown):

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

The interface above is parametrized by the specific type of belief base using Java Generics. All types of belief bases used within Tweety, such as propositional belief sets (`PlBeliefSet`) or probabilistic conditional knowledge bases (`PclBeliefSet`), are derived from `BeliefBase`. The package `net.sf.tweety.logics.commons.analysis` provides several generally applicable implementations of the above interface such as (only a simplified version is shown):

```
public class MiInconsistencyMeasure
  <S extends Formula,T extends BeliefSet<S>>
  implements InconsistencyMeasure<T> {

  private BeliefSetConsistencyTester<S,T>
      consTester;

  public MiInconsistencyMeasure(
      BeliefSetConsistencyTester<S,T>
      consTester){
    this.consTester = consTester;
  }

  @Override
  public Double inconsistencyMeasure
    (T beliefSet) {
      return new Double(this.consTester.
          minimalInconsistentSubsets(
          beliefSet).size());
  }
}
```

The above measure is an implementation of the MI-inconsistency measure [Grant and Hunter, 2006] and is applicable for all kinds of logics that provide an implementation of an `BeliefSetConsistencyTester`. This measure takes the number of minimal inconsistent subsets of a knowledge as an assessment of its inconsistency. Another example, particularly for the case of probabilistic conditional logic, is the `DistanceMinimizationInconsistencyMeasure` that makes use of the *Math* library. This measure assesses the grade of inconsistency by measuring how much the probabilities of the conditionals have to be modified in order to obtain a consistent knowledge base, cf. [Thimm, 2013b]. This problem is solved by optimization techniques that can be found in the *Math* library.

For probabilistic conditional logic, determining whether a knowledge base is inconsistent and assessing its inconsistency value is not easily done by hand. Using the implementations of various inconsistency measures in Tweety, we were able to compute and compare inconsistency values for various knowledge bases, cf. [Thimm, 2011]. Furthermore, as the interfaces and abstract classes provided by Tweety are very general and force the programmer to work as abstract as possible, even the implementation of such specific concepts such as inconsistency measures yield very generally applicable classes that are easily adapted to other approaches.

Strategic Argumentation

Our second case study is about strategic argumentation in multi-agent systems. In the works [Thimm and Garcia, 2010; Rienstra, Thimm, and Oren, 2013] we investigated systems of agents that are engaged in dialogues and aim at resolving contradiction by exchange of arguments. We give a brief introduction into the topic now, but simplify the formalization for the sake of readability.

We consider two agents PRO (proponent) and OPP (opponent) engaged in a dialogue about a specific argument A

(we use the terminology of abstract argumentation frameworks as mentioned earlier). The proponent has the goal to establish that A is acceptable and the opponent has the goal to establish that A is not acceptable. Both agents have only access to a subset of all available arguments and are, in general, ignorant or uncertain about the arguments the other agent has access to. Both agents take turn in forwarding a set of arguments. In [Rienstra, Thimm, and Oren, 2013] several different belief states with *opponent models* were proposed and discussed that help an agent to act strategically in these kinds of dialogues. The first type T_1 of belief state is a tuple (B, E) where B is the set of arguments a particular agent (either PRO or OPP) has access to, and E is the opponent model which is itself a belief state of type T_1[15]. This type of belief state therefore models what an agent thinks another agent beliefs, etc.. The second T_2 and third T_3 types of belief state extend the first type by introducing uncertainty on the set of arguments believed by the other agent and uncertainty about the arguments themselves. The second type of belief state T_2 is a tuple (B, P) where B is again the set of arguments a particular agent has access to and P is a probability distribution over some set $\{K_1, \ldots, K_n\}$ where each K_i $(i = 1, \ldots, n)$ is again a belief state of type T_2. For a formalization of the belief state of type T_3 see [Rienstra, Thimm, and Oren, 2013]. In [Rienstra, Thimm, and Oren, 2013] it has been analytically shown that the expressiveness of the three models is increasing from T_1 to T_3. However, in order to understand the differences between the three models examples have to be created and computed with different belief states. In the setting of strategic argumentation, this is a hard task to do by hand. In a system with at least two agents where both agents are equipped with a non-trivial belief state that changes with every action, running through a complete example by hand is a tedious task.

The complete setting of [Rienstra, Thimm, and Oren, 2013] has been implemented in the *Dialogues* library which makes heavy use of the general agent classes from the *Agents* library and, of course, the knowledge representation formalism from the *Abstract Argumentation* library. The central class of the implementation is the `ArguingAgent` class (we only show an excerpt):

```
public class ArguingAgent extends Agent {
  private BeliefState beliefState;
  private AgentFaction faction;

  @Override
  public Executable next(Collection<?
      extends Perceivable> percepts) {
    // [env = the environment object]
    this.beliefState.update(env.
        getDialogueTrace());
    return this.beliefState.move(env);
  }
}
```

The central attributes of an arguing agent are its belief state and its faction, e. g., either PRO or OPP. The method `next(...)` (derived from the super-class `Agent`) determines the agent's behavior on receiving some perception from the environment and returns some action (of type `Executable`). Here, the agent first updates its belief state with the current dialogue trace (a sequence of sets of arguments advanced so far) and then returns its own move (a set of arguments). The three different belief state types have been implemented in the classes `T1BeliefState`, `T2BeliefState`, and `T3BeliefState`. Arguing agents are organized in a `GroundedGameSystem` which is of type `MultiAgentSystem<ArguingAgent>` and models an argumentation dialogue ("grounded" refers to the grounded semantics used for this type of game). On top of this implementation of the actual dialogue system a simulation framework was implemented that allows the (random) generation of the above multi-agent systems and measures the performance of the individual agents over a series of runs. The central class for the simulation framework is the `GroundedGameGenerator` which implements the interface `MultiAgentSystemGenerator` and is able to generate (random) multi-agent systems of arguing agents.

In [Rienstra, Thimm, and Oren, 2013], for evaluating performance we generated a random abstract argumentation theory with 10 arguments, ensuring that the argument under consideration is in its grounded extension, i. e., under perfect information the proponent should win the dialogue. However, from these 10 arguments only $50\,\%$ are known by the proponent but $90\,\%$ by the opponent. We used a proponent without opponent model and generated an belief state of type T_3 for the opponent. From this T_3 belief state we derived T_2 and T_1 belief states by ignoring the added expressivity. For each belief state we simulated a dialogue against the same opponent and counted the number of wins. We repeated the experiment 5000 times, Figure 1 shows our results, cf. [Rienstra, Thimm, and Oren, 2013]. There, it can be seen that increasing the complexity of the belief state yields better overall performance (thus confirming the analytical evaluation). However, this empirical evaluation sheds also more light on the importance of the added expressivity for strategic argumentation. While T_2 is significantly better that T_1, the difference between T_3 and T_2 is nearly marginal. These kinds of nuances are very difficult to discover when considering only analytical evaluation.

5 Summary and Future Work

In this paper we presented Tweety, a comprehensive collection of Java libraries for logical aspects of artificial intelligence and knowledge representation. We gave an overview on the technical aspects and provided details on its individual packages. Finally, we presented two case studies that make use of Tweety as a framework for experimentation and empirical evaluation.

Tweety is an open source project[16,17] and can therefore be used and extended by everyone. In particular, instantiating the abstract Tweety classes for a particular formalism is sim-

[15]Note that this model has originally been proposed in [Oren and Norman, 2010]

[16]http://www.mthimm.de/projects/tweety/

[17]The source code of Tweety is hosted at SourceForge: http://tweety.svn.sourceforge.net.

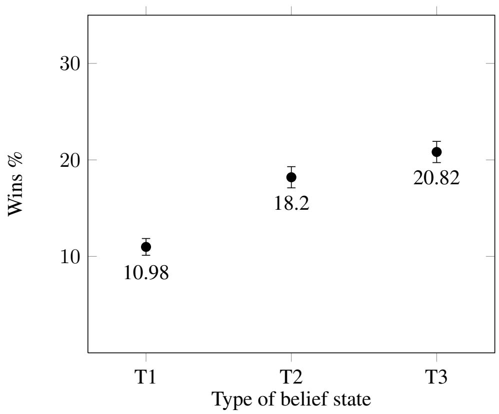

Figure 1: Average performance of T_1, T_2, and T_3 belief state models after 5000 simulation runs (with Binomial proportion confidence intervals)

ple. Although Tweety is implemented in a object-oriented programming language it follows a strict declarative formal way to define concepts from theoretical knowledge representation research. Tweety is available under the GNU General Public License version 3.0. In order to contribute to the main Tweety repository contact the author.

To the best of our knowledge, Tweety is the first attempt to provide a general-purpose framework for a broad variety of knowledge representation formalisms. However, there exist also more specialized frameworks for specific approaches or areas, such as the OWLAPI[18] for working with OWL ontologies, KReator[19] for relational probabilistic knowledge representation, or bcontractor[20] for belief dynamics.

Current and future work on Tweety is mainly concerned with extending the general infrastructure and improving usability. In particular, current work is about implementation of the plugin architecture for all libraries, a command line interface, and a web front-end. The ultimate goal there is to have several standardized user interfaces that are apt to work with any kind of knowledge representation mechanism and thus remove the burden of designing and implementing user interfaces from the researcher.

Acknowledgments

Tweety is being collaboratively developed by several contributors. Thanks go to Sebastian Homann, Tim Janus, Patrick Krümpelmann, Nico Potyka, Tjitze Rienstra, Stefan Tittel, Thomas Vengels, and Bastian Wolf. The research reported here was partially supported by the FP7 project SocialSensor (EC contract number 287975).

References

Antoniou, G., and van Harmelen, F. 2004. *A Semantic Web Primer*. Cambridge, MA, USA: MIT Press.

[18]http://owlapi.sourceforge.net

[19]http://kreator-ide.sourceforge.net

[20]https://code.google.com/p/bcontractor/

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press.

Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An introduction to argumentation semantics. *The Knowledge Engineering Review* 26(4):365–410.

Besnard, P., and Hunter, A. 2001. A logic-based theory of deductive arguments. *Artificial Intelligence* 128(1-2):203–235.

Besnard, P., and Hunter, A. 2006. Knowledgebase compilation for efficient logical argumentation. In *Proceedings of the 10th International Conference on Knowledge Representation (KR'06)*, 123–133. AAAI Press.

Brachman, R. J., and Levesque, H. J. 2004. *Knowledge Representation and Reasoning*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2007. A preference-based framework for updating logic programs. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*.

Delgrande, J. P. 1998. On first-order conditional logics. *Artificial Intelligence* 105(1-2):105–137.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence* 77(2):321–358.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about Knowledge*. MIT Press.

Fermé, E., and Hansson, S. O. 1999. Selective revision. *Studia Logica* 63(3):331–342.

Garcia, A., and Simari, G. R. 2004. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(1–2):95–138.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation - the a-prolog perspective. *Artificial Intelligence* 138(1-2):3–38.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electronic Transactions on AI* 2:193–210.

Goldszmidt, M., and Pearl, J. 1996. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence* 84:57–112.

Grant, J., and Hunter, A. 2006. Measuring inconsistency in knowledgebases. *Journal of Intelligent Information Systems* 27:159–184.

Hansson, S. O. 2001. *A Textbook of Belief Dynamics*. Norwell, MA, USA: Kluwer Academic Publishers.

Hunter, A., and Konieczny, S. 2006. Shapley inconsistency values. In *Proceedings of the 10th International Conference on Knowledge Representation (KR'06)*, 249–259. AAAI Press.

Karunatillake, N. C.; Jennings, N. R.; Rahwan, I.; and McBurney, P. 2009. Dialogue games that agents play within a society. *Artificial Intelligence* 173(9-10):935–981.

Kern-Isberner, G., and Thimm, M. 2010. Novel semantical approaches to relational probabilistic conditionals. In Lin, F.; Sattler, U.; and Truszczyński, M., eds., *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR'10)*, 382–392. AAAI Press.

Kern-Isberner, G., and Thimm, M. 2012. A ranking semantics for first-order conditionals. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*.

Kern-Isberner, G. 2001. *Conditionals in Nonmonotonic Reasoning and Belief Revision*. Number 2087 in Lecture Notes in Computer Science. Springer-Verlag.

Krümpelmann, P., and Kern-Isberner, G. 2012. Belief base change operations for answer set programming. In Cerro, L. F.; Herzig, A.; and Mengin, J., eds., *Proceedings of the 13th European conference on Logics in Artificial Intelligence (JELIA'12)*, 294–306. Springer.

Krümpelmann, P.; Thimm, M.; Falappa, M. A.; Garcia, A. J.; Kern-Isberner, G.; and Simari, G. R. 2011. Selective revision by deductive argumentation. In Modgil, S.; Oren, N.; and Toni, F., eds., *Theory and Applications of Formal Argumentation, Proceedings of the First International Workshop (TAFA'11, revised selected papers)*, volume 7132 of *Lecture Notes in Artificial Intelligence*, 147–162. Springer.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence* 25:369–389.

McCabe, T. J. 1976. A complexity measure. *IEEE Transactions on Software Engineering* 308–320.

Nute, D., and Cross, C. 2002. Conditional logic. In Gabbay, D., and Guenther, F., eds., *Handbook of Philosophical Logic*, volume 4. Kluwer Academic Publishers, second edition edition. 1–98.

Oren, N., and Norman, T. J. 2010. Arguing using opponent models. In *Proceedings of the 6th international conference on Argumentation in Multi-Agent Systems*, ArgMAS'09, 160–174. Berlin, Heidelberg: Springer-Verlag.

Paris, J. B. 1994. *The Uncertain Reasoner's Companion – A Mathematical Perspective*. Cambridge University Press.

Pearl, J. 1990. System Z: a natural ordering of defaults with tractable applications to default reasoning. In *Proceedings of the 3rd conference on Theoretical aspects of reasoning about knowledge (TARK'90)*, 121–135. Morgan Kaufmann Publishers.

Potyka, N. 2014. Linear programs for measuring inconsistency in probabilistic logics. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1–2):107–136.

Rienstra, T.; Thimm, M.; and Oren, N. 2013. Opponent models with uncertainty for strategic argumentation. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*.

Rödder, W. 2000. Conditional logic and the principle of entropy. *Artificial Intelligence* 117:83–106.

Schweimeier, R., and Schroeder, M. 2003. A parameterised hierarchy of argumentation semantics for extended logic programming and its application to the well-founded semantics. *Theory and Practice of Logic Programming* 5(1–2):207–242.

Spohn, W. 1988. Ordinal conditional functions: a dynamic theory of epistemic states. In Harper, W., and Skyrms, B., eds., *Causation in Decision, Belief Change, and Statistics*, volume 2. Kluwer Academic Publishers. 105–134.

Stolzenburg, F.; Garcia, A.; Chesnevar, C. I.; and Simari, G. R. 2003. Computing generalized specificity. *Journal of Non-Classical Logics* 13(1):87–113.

Thimm, M., and Garcia, A. J. 2010. Classification and Strategical Issues of Argumentation Games on Structured Argumentation Frameworks. In van der Hoek, W.; Kaminka, G. A.; Lespérance, Y.; Luck, M.; and Sen, S., eds., *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems 2010 (AAMAS'10)*.

Thimm, M., and Kern-Isberner, G. 2008. A distributed argumentation framework using defeasible logic programming. In Besnard, P.; Doutre, S.; and Hunter, A., eds., *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA'08)*, number 172 in Frontiers in Artificial Intelligence and Applications, 381–392. IOS Press.

Thimm, M. 2011. *Probabilistic Reasoning with Incomplete and Inconsistent Beliefs*, volume 331 of *Dissertations in Artificial Intelligence*. IOS Press.

Thimm, M. 2012. A probabilistic semantics for abstract argumentation. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*.

Thimm, M. 2013a. Dynamic preference aggregation under preference changes. In *Proceedings of the Fourth Workshop on Dynamics of Knowledge and Belief (DKB'13)*.

Thimm, M. 2013b. Inconsistency measures for probabilistic logics. *Artificial Intelligence* 197:1–24.

Walsh, T. 2007. Representing and reasoning with preferences. *AI Magazine* 28(4):59–70.

Weiss, G., ed. 2013. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, second edition.