The making of SPINdle*

Ho-Pun Lam[1,2] and Guido Governatori[2]

[1] School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Australia
[2] NICTA, Queensland Research Laboratory, Brisbane, Australia

Abstract. We present the design and implementation of SPINdle – an open source Java based defeasible logic reasoner capable to perform efficient and scalable reasoning on defeasible logic theories (including theories with over 1 million rules). The implementation covers both the standard and modal extensions to defeasible logics. It can be used as a standalone theory prover and can be embedded into any applications as a defeasible logic rule engine. It allows users or agents to issues queries, on a given knowledge base or a theory generated on the fly by other applications, and automatically produces the conclusions of its consequences. The theory can also be represented using XML.

Key words: Defeasible Logic, Modal Defeasible Logic, Reasoning

1 Introduction

Defeasible logic (DL) is a non-monotonic formalism originally proposed by Nute [1]. It is a simple rule-based reasoning approach that can reason with incomplete and contradictory information while preserving low computational complexity [2]. Over the years, the logic has been developed notably by [3,4,5]. Its use has been advocated in various application domains, such as business rules and regulations [6], agent modeling and agent negotiations [7], applications to the Semantic Web [8] and business process compliance [9]. It is suitable to model situations where conflicting rules may appear simultaneously.

In this paper we report on the implementation of SPINdle which implements reasoners to compute the consequences of theories in defeasible logic. The implementation covers both standard defeasible logic and modal defeasible logic.

The most important features of SPINdle are the following:

- It supports all rule types of defeasible logic, such as fact, strict rules, defeasible rules, defeaters and superiority.
- It supports Modal Defeasible Logics [10] with modal operator conversions.
- It supports negation and conflicting (mutually exclusive) literals.
- A theory can be represented using XML and plain text (with pre-defined syntax), and a theory and its extension can also be exported using XML.

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

G. Governatori, J. Hall and A. Paschke (eds)
Rule Representation, Interchange and Reasoning on the Web, RuleML 2009
LNCS 5858, 2009.

– A visual theory editor is developed for editing standard defeasible logic theory.

As a result, SPINdle is a powerful tools supporting:

– rule, facts and ontologies;
– monotonic and nonmonotonic (modal) rules reasoning with inconsistencies and incomplete information

In the rest of this paper, section 2 gives a brief introduction to the syntax and semantics of both standard defeasible logic and modal defeasible logic. Section 3 describes the implementation details of SPINdle, the algorithm that it used and the data structure that it proposed to enhance the performance of the inference process. Section 4 presents the performance statistics from a study undertaken using various types and sizes of defeasible logic theories. Section 5 gives our conclusions and poses future research/development work on SPINdle.

2 Defeasible Logic

2.1 Basics of Defeasible Logic

A *defeasible theory D* is a triple $(F,R,>)$ where F and R are finite set of facts and rules respectively, and $>$ is a superiority relation on R. Here SPINdle only considers rules that are essentially propositional. Rules containing free variables are interpreted as the set of their ground instances.

Facts are indisputable statements, represented either in form of states of affairs (literal and modal literal) or actions that have been performed. Facts are represented by predicates. For example, "John is a human" is represented by $human(John)$.

A *rule*, on the other hand, describes the relations between a set of literals (premises) and a literal (conclusion). We can specify the strength of the rule relation using the three kinds of rules supported by DL, namely: *strict rules*, *defeasible rules* and *defeaters*; and can specify the mode the rules used to connect the antecedent and the conclusion. However, in such situations, the conclusions derived will be *modal literals*.

Strict rules are rules in the classical sense: whenever the premises are indisputable (e.g. facts) then so is the conclusion. An example of a strict rule is: "human are mammal", written formally:

$$human(X) \rightarrow mammal(X)$$

Defeasible rules are rules that can be defeated by contrary evidence. An example of such a rule is "mammal cannot flies"; written formally:

$$mammal(X) \Rightarrow \neg flies(X)$$

The idea is that if we know that X is a mammal then we may conclude that it cannot fly *unless there is other, not inferior, evidence suggesting that it may fly* (for example that mammal is a bat).

Defeaters are a special kind of rules that cannot be used to draw any conclusions. Their only use is to prevent some conclusions. That is, they are used to defeat some defeasible rules by producing evidence to the contrary. For example the rule

$$heavy(X) \rightsquigarrow \neg flies(X)$$

states that an animal is heavy is not sufficient enough to conclude that it does not fly. It is only evidence against the conclusion that a heavy animal flies. In other words, we don't wish to conclude that $\neg$*flies* if *heavy*, we simply want to prevent a conclusion *flies*.

DL is a "skeptical" nonmonotonic logic, meaning that it does not support contradictory conclusions. Instead DL seeks to resolve conflicts. In cases where there is some support for concluding A but also support for concluding $\neg A$, DL does not conclude neither of them. However, if the support for A has priority over the support for $\neg A$ then A is concluded.

As we have alluded to above, no conclusion can be drawn from conflicting rules in DL unless these rules are prioritised. The *superiority relation* is used to define priorities among rules, that is, where one rule may override the conclusion of another rule. For example, given the following facts:

$$\rightarrow bird \qquad\qquad \rightarrow brokenWing$$

and the defeasible rules:

$$\begin{array}{rr} r: & bird \Rightarrow fly \\ r': & brokenWing \Rightarrow \neg fly \end{array}$$

which contradict one another, no conclusion can be made about whether a bird with a broken wing can fly. But if we introduce a superiority relation $>$ with $r' > r$, then we can indeed conclude that the bird cannot fly.

We now give a short informal presentation of how conclusions are drawn in DL. Let D be a theory in DL (as described above). A *conclusion* of D is a tagged literal and can have one of the following four forms:
$+\Delta q$ meaning that q is definitely provable in D (i.e. using only facts and strict rules).
$-\Delta q$ meaning that we have proved that q is not definitely provable in D.
$+\partial q$ meaning that q is defeasible provable in D.
$-\partial q$ meaning that we have proved that q is not defeasible provable in D.

Strict derivations are obtained by forward chaining of strict rules while a defeasible conclusion p can be derived if there is a rule whose conclusion is p, whose prerequisites (antecedent) have either already been proved or given in the case at hand (i.e. facts), and any stronger rule whose conclusion is $\neg p$ has prerequisites that fail to be derived. In other words, a conclusion p is (defeasibly) derivable when:

- p is a fact; or
- there is an applicable strict or defeasible rule for p, and either
  - all the rules for $\neg p$ are discarded (i.e. not applicable) or
  - every rule for $\neg p$ is weaker than an applicable rule for p.

A full definition of the proof theory can be found in [3]. Roughly, the rules with conclusion p form a team that competes with the team consisting of the rules with conclusion $\neg p$. If the former team wins p is defeasibly provable, whereas if the opposing team wins, p is non-provable.

2.2 Modal Defeasible Logic

Modal logics have been put forward to capture many different notions somehow related to the intensional nature of agency as well as many other notions. Usually modal logics

are extensions of classical propositional logic with some intensional operators. Thus any modal logic should account for two components: (1) the underlying logical structure of the propositional base and (2) the logical behavior of the modal operators. Alas, as is well-known, classical propositional logic is not well suited to deal with real life scenarios. The main reason is that the descriptions of real-life cases are, very often, partial and somewhat unreliable. In such circumstances, classical propositional logic is doomed to suffer from the same problems.

On the other hand, the logic should specify how modalities can be introduced and manipulated. Some common rules for modalities are, e.g., **Necessitation** (from $\vdash \phi$ infer $\vdash \Box\phi$) and **RM** (from $\vdash \phi \rightarrow \psi$ infer $\vdash \Box\phi \rightarrow \Box\psi$). Both dictate conditions to introduce modalities purely based on the derivability and structure of the antecedent. These rules are related to well-known problem of omniscience and put unrealistic assumptions on the capability of an agent. However, if we take a constructive interpretation, we have that if an agent can build a derivation of ψ then she can build a derivation of $\Box\psi$.

To this end, SPINdle follows the semantics proposed by [10] on reasoning with modal defeasible logic. However, due to the limited space, readers interested in understand the semantics, modal operator conversions, conflict detections, conflict resolutions, and algorithm implemented in SPINdle please refer to the paper for details.

3 Implementation

3.1 SPINdle System Architecture

SPINdle, written in Java, consists of three major components (Fig. 1): the Rule Parser, the Theory Normalizer and the Inference Engine.

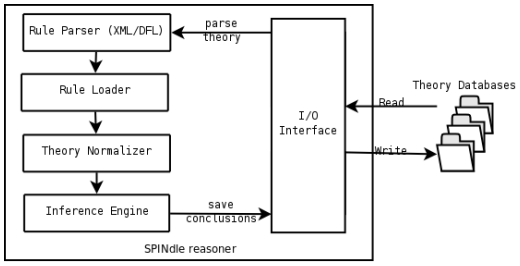


Fig. 1: Main components of the SPINdle reasoner.

SPINdle accepts defeasible logic theories represented using XML or plain text (with pre-defined syntax). The *Rule Parser* is used to transform the theory from a saved theory document into a data structure that can be processed in the next module. The nature of the rule parser is rather similar to the *Logic Program Loader* module of the DR-Device family of applications developed by [8].

After loading a theory into SPINdle, users can then modify/manipulate the theory according to their applications need. The *I/O Interface* module in Fig. 1 provides useful

methods for helping users to load and save (modified) theory (also the derived conclusions) to and from the database. Theories can also be exported using XML for agent communication, which is a common scenario in the Semantic Web.

3.2 The Inference process

The whole inference process has two phases: A *pre-processing* phase where we transform the theory using the techniques described in [3] into an equivalent theory *without superiority relation and defeaters*, which later helps to simplify the reasoning process in the reasoning engine. The *Theory Normalizer* module in Fig. 1 carry out this process by further breaking it down into three linear transformations: one to transform the theory to regular form, one to empty the superiority relation and one to empty the defeaters. In addition to this, the theory normalizer also transform rules with *multiple heads* into an equivalent sets of rules with single head. It is expected that the transformed theory will produce the same sets of conclusions in the language of the theory they transform.

Following [2] (a.k.a. the *Delores* algorithm), the *conclusions generation* phase (the *Inference Engine* module) is based on a series of (theory) transformations that allow us (1) to assert whether a literal is provable or not (and the strength of its derivation); (2) to progressively reduce and simplify a theory. The reasoner will, in turn:

- Assert each fact (as an atom) as a conclusion and remove the atom from the rules where the atom occurs positively in the body, and "deactivates" (remove) the rule(s) where the atom occurs negatively in the body. The atom is then removed from the list of atoms.
- Scan the list of rules for rules with empty head. It takes the head element as an atom and search for rule(s) with conflicting head. If there are no such rules then the atom is appended to the list of facts and the rule will be removed from the theory. Otherwise the atom will append to the pending conclusion list until all rule(s) with conflicting head can be proved negatively.
- Repeats the first step.
- The algorithm terminates when one of the two steps fails or when the list of facts is empty. On termination the reasoner output the set of conclusions.

Finally, the conclusions are either exported to the users, or saved in the theory database for future use. Since each atom/literal in a theory is processed exactly once and every time we have to scan the set of rules, the complexity of the above algorithm is $O(|\mathscr{L}| * |\mathscr{R}|)$, where $\mathscr{L}$ is the size of the distinct modal literals and $\mathscr{R}$ is the number of rules in the theory [10]. This complexity result can be improved through the use of proper data structure (Fig. 2). For each literal p a linked-list (the dashed arrow) of the occurrences of p in rule(s) can be created during the theory paring phase. Each literal occurrence has a link to the record for the rule(s) it occurs in. Using this data structure, whenever a literal update occurs, the list of rules relevant to that literal can be retrieved easily. Thus instead of scanning through all the rules for empty head (step 2), only rules relevant to that particular literal will be scanned. The complexity of the inference process will therefore reduced to $O(|\mathscr{L}| * |\mathscr{M}|)$, where $\mathscr{M}$ is the maximum number of rules a literal associated with in the theory,

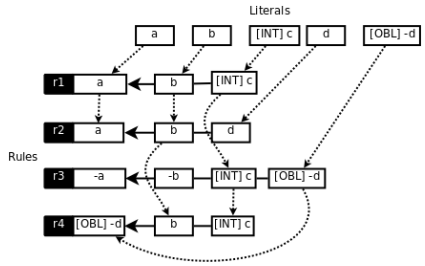


Fig. 2: Data structure for literals-rules association.

It is important to note that the aforementioned algorithm is a generalized version that is common in inferencing both standard defeasible logic and modal defeasible logic. In the case of modal defeasible logic, due to the modal operator conversions, an additional process adding extra rules to the theory is needed. In addition, for a literal p with modal operator $\Box_i$, besides its complement (i.e., $\Box_i \neg p$), the same literal with modal operator(s) in conflict with $\Box_i$ should also be included in the conflict literal list (step 2) and only literal with strongest modality will be concluded.

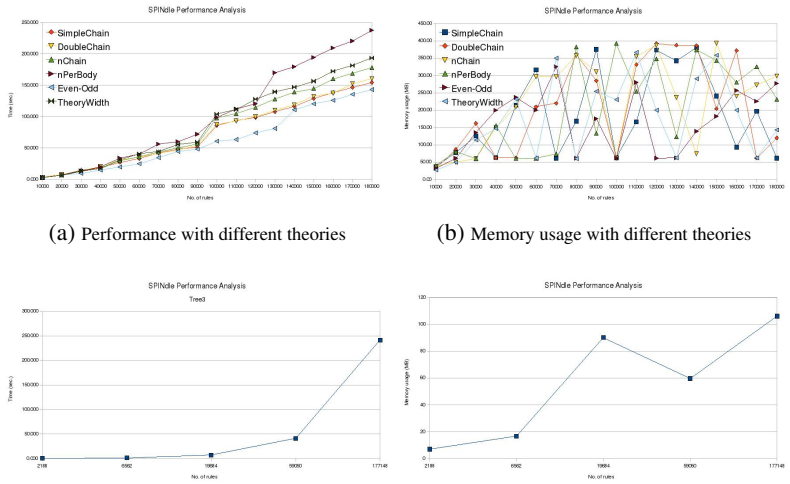
4 Performance Evaluation

SPINdle has been extensively tested for correctness, scalability and performance using different forms of theories generated by a tool that comes with Deimos [11]. In this section we are going to describe the tests we have performed and the result we obtained.

In the test, SPINdle is compiled using the SUN Java SDK 1.6 without any optimization flags. The test begins with a heap size of 512MB and gradually increased to 2GB according to the theory size. Performance has been measured on a Pentium 4 PC (3GHz) with Linux (Ubuntu) and 2GB main memory. Figure 3 shows the performance and memory usage of SPINdle in running theories of various types and sizes.

The various theory types were designed to explore various aspects of defeasible inference. For example, in theory simple-chain of size n, a fact a_0 is at the end of a chain of n rules $a_i \Rightarrow a_{i+1}$. A defeasible proof of a_n will use all the rules and the fact. In theory *tree(n,k)*, a_0 is the root of a k-branching tree of depth n, in which every literals occur once. More details about the various theory types can be found in [11]. Notices that in our performance evaluation the theory sizes refer to the number of rules (both strict rules and defeasible rules) that were stored in the theory. The statistics show only the time used for the inference process (excluding the time used for loading theories into the reasoner) and the total memory used. Figure 3 (a,b) show the performance and memory usage with large theory ($10000 < n < 180000$)[3]; while (c,d) show the same information in theory *tree* with $k = 3$ and n=$\{2188, 6562, 19684, 59050$ and $177148\}$.

[3] SPINdle does not impose any limitation on the theory size. SPINdle can reason on any theory as long as it can be put onto the memory. The largest theory size tested so far is with $n = 1,000,000$ (1 million).



(a) Performance with different theories

(b) Memory usage with different theories

(c) Performance in theory *Tree3*

(d) Memory usage in theory *Tree3*

Fig. 3: Performance and Memory usage with different theories

Results (Fig. 3) show that SPINdle can handle inferences with thousands of rules in less than three seconds. The reasoning time growth (in most cases) almost linearly proportional to the size of the theories tested, which is coherent with the complexity described in section 3.2.

On the contrary, the memory usage fluctuates as the theories sizes increase, regardless the semantics of the input theories. This may be due to the dynamic nature of hash table that we used in handling the literal and theory objects as well as the temporary variables that we created during the inference process. Nevertheless, for the theories tested, our results show that SPINdle can handle 10,000 rules with less than 50MB memory and 180,000 rules with less than 400MB.

Last but not least, apart from the default configuration, we can also optimize the performance (also the memory usage) of SPINdle through different setting. For example, a *defeasible rule only* engine can be used (instead of the ordinary one) if the input theory contains defeasible rules only. Our results (not shown here) show that the performance can be improved by up to 3% and with about 20% drop in memory usage (on average) if we configure SPINdle properly.

5 Conclusion

We have presented SPINdle, a reasoner that support reasoning on both the standard defeasible logic and modal defeasible logic. It features include:

- Full implementation of defeasible logic, including fact, strict and defeasible rules, defeaters, and superiority relation among rules.
- It supports reasoning with Modal Defeasible Logic theory.

- It can reason with incomplete and inconsistent theories.
- It is efficient (due to the low computational complexity), and with low memory consumption.
- It supports theory represented in both XML and pre-defined plain text format. Theory can also be exported using XML for agent communication.

We have also showed that the complexity of the inference process can be reduced through the use of proper data structure.

SPINdle is freely available for downloading and experimentation under the LGPL license agreement, at the following address:

`http://spin.nicta.org.au/spindleOnline`

To further enhance the usability of SPINdle, a visual editor for writing rules and theory in standard defeasible logic was developed and is also available for download.

Future directions for SPINdle include both algorithm and technological improvement, which include interface to support direct import from the web and processing of OWL/RDF data and RDF schema ontologies. Reasoning support for *Temporal Defeasible Logic* [12] will also be included in the future.

References

1. Nute, D.: Defeasible logic. In Gabbay, D., Hogger, C., eds.: Handbook of Logic for Artificial Intelligence and Logic Programming. Volume III. Oxford University Press (1994) 353–395
2. Maher, M.J.: Propositional defeasible logic has linear complexity. Theory and Practice of Logic Programming **1**(6) (2001) 691–711
3. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Transactions on Computational Logic **2**(2) (2001) 255–286
4. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Embedding defeasible logic into logic programming. Theory and Practice of Logic Programming **6**(6) (2006) 703–735
5. Billington, D.: Defeasible logic is stable. J Logic Computation **3**(4) (1993) 379–400
6. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: On the modeling and analysis of regulations. In: Proceedings of the Australian Conference Information Systems. (1999) 20–29
7. Governatori, G., Rotolo, A.: Defeasible logic: Agency, intention and obligation. In Lomuscio, A., Nute, D., eds.: Deontic Logic in Computer Science. Number 3065 in LNAI, Springer-Verlag (2004) 114–128
8. Bassiliades, N., Antoniou, G., Vlahavas, I.: A defeasible logic reasoner for the semantic web. International Journal of Semantic Web and Information Systems **2**(1) (2006) 1–41
9. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: 10th International Enterprise Distributed Object Computing Conference (EDOC 2006), IEEE Computing Society (2006) 221–232
10. Governatori, G., Rotolo, A.: BIO logical agents: Norms, beliefs, intentions in defeasible logic. Journal of Autonomous Agents and Multi Agent Systems **17** (2008) 36–69
11. Maher, M.J., Rock, A., Antoniou, G., Billington, D., Miller, T.: Efficient defeasible reasoning systems. International Journal of Artificial Intelligence Tools **10** (2001) 483–501
12. Governatori, G., Rotolo, A., Sartor, G.: Temporalised normative positions in defeasible logic. In: 10th International Conference on Artificial Intelligence and Law (ICAIL05), ACM Press (2005) 25–34