On Truth Maintenance with Literals.

Wiktor Marek[1] and Miroslaw Truszczynski[2]

1 Introduction

By a truth-maintenance system (TMS) over a collection of atoms At, we mean a collection of *justification rules* of form $r =< A|B \rightarrow c >$, where $A, B \subseteq At$ and $c \in At$. Such collection of rules $\mathcal{S}$ can be assigned an TMS-extension (c.f. [Dr88]). Such extension may be represented either by means of objects that possess a derivation (c.f. [MT89], [RDB89]) or by means of a nonmonotonic operator, essentially due to Reiter ([Re80]) as follows:

$$\Gamma(Z) = \bigcap\{U\colon \forall_{r\in\mathcal{S}}(((r =< A|B \rightarrow c >) \wedge A \subseteq U \wedge B \cap Z = \emptyset) \Rightarrow c \in U))\}$$

One can prove that Z is an TMS-extension of $\mathcal{S}$ if and only if $\Gamma(Z) = Z$. In particular this definition is equivalent to one given in [RDB89], where an extension is characterized as the collection of atoms with the groundedness property and closed under derivation. This, in tutn, is equivalent to one given, for default logic, in [MT89]

[1]Department of Computer Science, University of Kentucky, Lexington, KY 40506, currently with Mathematical Sciences Institute, Cornell University

[2]Department of Computer Science, University of Kentucky, Lexington, KY 400506

where an extension is characterized by means of fixpoints of parametrized monotone operators (with the condition that fixpoint is identical with the parameter). Recently, Gelfond and Lifschitz in [GL89] indicated how to introduce classical negation into logic programming. Here, modifying their approach by means of introduction of *inconsistency spreading rules*, we show how to extend the construction of the TMS extensions to the situation when - instead of atoms - we deal with literals.

2 Construction

Let At be a collection of atoms. A corresponding collection of literals Lit_{At} (denoted below by Lit) consists of signed objects $< \varepsilon, a >$, where $a \in At$, $\varepsilon \in \{0,1\}$. Customarily we abbreviate $< o, a >$ as a and $< 1, a >$ as $\neg a$.

A collection of literals C is *inconsistent* if for some $a \in At$, both a and $\neg a$ belong to C. Otherwise it is called *consistent*.

We introduce now the notion of a justification rule in the same way as it was done above, except that now we allow A, B to be subsets of Lit rather than At and c belonging to Lit.

If we do so, however, and proceed with definition of Γ above, some unexpected phenomenon happens:

Example 2.1 *Let $\mathcal{S}$ be the following system: $< \mid > \rightarrow a$, $< \mid > \rightarrow c$, $< a \mid > \rightarrow d$, $< c \mid b > \rightarrow e$, $< a \mid f > \rightarrow \neg d$.*

With the underlying collection of atoms consisting of $\{a, , b, c, d, e, f\}$.

It is easy to see that the collection $C = \{a, c, d, \neg d, e\}$ satisfies definition of extension. Clearly C is inconsistent. Yet, in opposition to the intuition from logic, namely that an inconsistent collection entails everything, here an inconsistent collection does not entail every literal.

Some may find this situation appealing, arguing that the inconsistency with respect to an atom a witnesses only to incoherence of a part of the system (see for instance [BS90] for a specific examples and an argument). We adopt here an opposite position, namely that once an inconsistency has been detected, every literal follow. In the logic programming context the same position is taken by [GL89].

Thus, instead of accepting a *partial inconsistency* like one in our example, or simply arbitrary assigning to $\mathcal{S}$ the collection Lit as an extension (cf [GL89]) we introduce additional processing rules, called *inconsistency spreading rules* of form:

$$< \{a, \neg a\} \mid > \rightarrow c$$

For every $a \in At$ and $c \in Lit$. Let ISR_{Lit} be the collection of inconsistency spreading rules associated with Lit. We have the following theorem:

Theorem 2.1 *Let $\mathcal{S}$ be a truth maintenance system with the collection of literals Lit. Let C be a consistent subset of Lit. Then C is an extension of $\mathcal{S}$ if and only if C is an extension of $\mathcal{S} \cup ISR_{Lit}$.*

Proof: Consider Γ_1, Γ_2 the Reiter operators for $\mathcal{S}$, and $\mathcal{S} \cup ISR_{Lit}$ respectively. We show that consistent fixpoints of Γ_1 and Γ_2 coincide.

(a) Assume $\Gamma_1(R) = R$, and R consistent. Then R is closed under rules from ISR_{Lit}. These rules have no negative part and so their applicability does not depend negatively on R at all. Now, the inclusion $\Gamma_2(R) \subseteq \Gamma_1(R)$ is obvious. As concerns the converse inclusion $\Gamma_1(R) \subseteq \Gamma_2(R)$, notice that the family whose intersection is $\Gamma_2(R)$ may have only one more object, namely all Lit. But all the sets in $\Gamma_2(R)$ are included in Lit, so the intersection is preserved.

Implication $\Gamma_2(R) = R$ and R consistent implies $\Gamma_1(R) = R$ follows a similar line. $\square$

Proposition 2.2 *If C is an extension of $\mathcal{S} \cup ISR_{Lit}$ then C is inconsistent if and only if $C = Lit$.*

Proof: Only the implication $\Leftarrow$ is nontrivial, and follows immediately from the effect of closure C under ISR.

The construction we gave equally applies to logic programming with classical negation. Here the collection of *inconsistency spreading clauses*, ISC, takes form:

$$a \leftarrow q, \neg q$$

for every literal a and atom q. Notice that negation $\neg$ is different from additional epistemic negation *not*. We have then the following:

Proposition 2.3 *Let P be a logic program with classical negation. Then a consistent set of literals is an answer set of P if and only if it is an answer set of $P \cup ISC$.*

Thus we have the following choice while processing: we can either adopt additional clauses ISC or else introduce additional atoms which code negative literals.

References

[BS90] C. Baral and V.S. Subrahmanian Stable Classes for Logic Programs. Unpublished Notes.

[Do80] J. Doyle. A Truth Maintenance System. *Artificial Intelligence Journal* 12:231–272, 1979.

[Dr88] O. Dressler. An Extended Basic ATMS In: *Non-Monotonic Reasoning Lecture Notes in Computer Science* 346:143-163, Springer Verlag, 1988.

[Kl86] J. de Kleer. An Assumption-based TMS. *Artificial Intelligence* 28:127 – 162, 1986.

[GL88] M. Gelfond, V. Lifschitz. The Stable Model Semantics for Logic Programming. In: R. Kowalski and K. Bowen, eds., *Proceedings of 5th International Symposium Conference on Logic Programming*, Seattle, 1988.

[GL89] M. Gelfond, V. Lifschitz. Logic Programming with Classical Negation. Unpublished Notes.

[MT89] W. Marek and M. Truszczyński. Relating Autoepistemic and Default Logics, In: *Principles of Knowledge Representation and Reasoning*, pages 276 – 288, Morgan Kaufman, San Mateo, 1989.

[RDB89] M. Reinfrank, O. Dressler and G. Brewka. On the Relation between Truth Maintenance and Non-Monotonic Logics *Proceedings of IJCAI*, 1989.

[Re80] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence Journal*, 13:81–132, 1980.