Properties of maximal cliques of a pair-wise compatibility graph for three nonmonotonic reasoning systems

R. E. Mercer[1] and V. Risch[2]

[1] Cognitive Engineering Laboratory, Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7
[2] InCA Team, LSIS - UMR CNRS 6168, Domaine Universitaire de Saint-Jérôme, avenue Escadrille Normandie Niemen, 13397 Marseille cédex 20, France

Abstract. In this paper we define the notion of a compatibility relation so as to have a common framework for three nonmonotonic reasoning systems: normal logic programming, extended logic programming, and a restricted form of default logic. We show some properties of the maximal cliques of the pair-wise compatibility graph givn by the relation between the rules of the various reasoning systems. Properties that these maximal cliques possess are presented. A procedure to compute stable models (resp. answer sets, extensions) by enumerating maximal cliques with intelligent pruning is sketched.

Keywords: Logic Programming, Default Logic, pair-wise compatibility relation, compatibility graph, maximal cliques

1 Introduction

Common features of three nonmonotonic reasoning systems, normal logic programs, extended logic programs, and Reiter's default reasoning, have been well-studied [10]. We present a new common framework for these three nonmonotonic reasoning systems. In the sequel we call this common framework *the computational pipeline*.

Fundamental to the computational pipeline is the compatibility graph. The graph structure represents pair-wise rule compatibility, that is, a binary relation indicating the two rules do not conflict, given only the information contained in the two rules. Maximal cliques of a compatibility graph represent the only subsets of rules that need to be considered as potentially contributing to the computation of stable models (resp. answer sets, Reiter-extensions).

The purpose of the common framework is to provide a means to investigate common properties of the computational pipeline in each of the three systems in terms of properties of the compatibility graph maximal cliques. A number of properties of the maximal cliques of the compatibility graph are presented: the cliques represent subsets of the original rules; each clique represents a single Horn theory, so it contains at most, and in many cases exactly one stable model (resp. answer set, extension); the compatibility graph can be computed in $O(n^2)$ (with small coefficients) time and requires $O(n^2)$ space; and because polynomial-delay algorithms exist to compute maximal cliques, the stable models (resp. answer sets, extension), including the first, for certain classes of logic programs (resp. default theories) can be computed with polynomial delay.

Anticipating the negative computational properties of the simple view of the computational pipeline ($O(2^n)$ maximal cliques, and the potential multiple generation of stable models (resp. answer sets, extensions)), we sketch a method that views the computational pipeline as enumeration of maximal cliques using a tree structure with pruning done at any node in the tree, rather than only at leaf nodes which the pipeline structure possibly suggests. This method incorporates the relaxed stratification view of rule interaction [2].

2 Preliminaries

A *graph* is a pair (V, E) where V is a set and E is a subset of $V \times V$. If $V \neq \emptyset$ the graph is *non-empty*. Elements in V are called *nodes* and members of E are called *edges*. Given a graph $G = (V, E)$, a subgraph C =(V',E'), where $V' \subseteq V$, $E' \subseteq E$, is *a maximal clique of* G iff (i) $(\forall v_i v_j \in V')((v_i, v_j) \in E')$ and (ii) $(\forall v_i \notin V')(\exists v_j \in V')((v_i, v_j) \notin E)$ Throughout this paper, we refer to maximal cliques as *cliques*.

In the sequel, we consider graphs made of rules (either *default theories* or rules of *normal logic programs* and *extended logic programs*). We consider these concepts familiar and refer the reader to the basic sources on the subject [16] [10]. Note that throughout this paper, we consider only a restricted form of propositional default theories[1] where (1) the language has no disjunction except in the special case of horn clauses in W, (2) W contains no conjunctions, (3) the consequence and each justification is an atom, and (4) W is emptied by the following transformation: facts are transformed into prerequisite-free, justification-free rules (or in the sense of logic programs, bodiless rules) and horn clauses are transformed into justification-free rules (or rules without 'not's in the logic program sense). This leads us to use the language of atoms for normal logic programs and the language of atoms with classical negation in the case of default theories and extended logic programs. Hence, we abuse the use of the generic term "atom" to include atoms with classical negation, considering that the context will make clear what is meant. We use the following notions (most of them developed in [13] and [15]): Atoms and not-atoms (formulas of the form not(a)) are called *literals*. For a set of literals S, by S^+ (resp. S^-) we denote the set of atoms (resp. not-atoms) in S. Moreover, we consider $|S| = \{a \mid \text{not}(a) \in S\}$ and $\neg S = \{\neg a \mid a \in S^+\}$. Hence, logic programs consist of rules of the form

$$\textit{head} \leftarrow \textit{body}^+, \textit{body}^-$$

corresponding modularly to default rules of the form

$$\frac{\bigwedge \textit{body}^+ : \neg\, |\textit{body}^-|}{\textit{head}}$$

where *head* is an atom and *body* is a set of literals (cf. [7]). $\textit{NAnt}(P)$, called the *negative antecedents* in P, is the set of atoms a such that not(a) appears in P, where

[1] More general properties of the pair-wise compatibility graph have been proven for general default reasoning in [12]

P is a logic program, i.e. $\mathit{NAnt}(P) = \bigcup_{r \in P} |\mathit{body}^-|$. The *deductive closure* of a set of ground rules P of a logic program (resp. of a set of defaults) and a set of literals L, denoted $\mathit{Dcl}(P, L)$, is the smallest set of atoms which contains L^+ and is closed under the inference rules $R(P, L)$ where $R(P, L) = \{\text{head} \leftarrow \text{body}^+ \mid \text{head} \leftarrow \text{body}^+, \text{body}^- \in P,\ \text{body}^- \subseteq L^-\}$. As stressed in [15], Δ is a *stable model* of a set of ground rules P iff $\Delta = \mathit{Dcl}(P, \mathit{NAnt}(P) - \Delta)$. There is the immediate counterpart in terms of extensions of a default theory. A set Λ of not-atoms is called *P-full* iff $(\forall a)(a \in \mathit{NAnt}(P))(\text{not}(a) \in \Lambda \mathit{\ iff\ } a \notin \mathit{Dcl}(P, \Lambda))$. Finally, note the following result:

Theorem 1. *(Theorem 2.2 of [15]) (i) If Λ is P-full, then $\mathit{Dcl}(P, \Lambda)$ is a stable model of a set of ground rules P. (ii) If there is a stable model of Δ of P, then $\Lambda = \text{not}(\mathit{NAnt}(P) - \Delta)$ is a P-full set such that $\Delta = \mathit{Dcl}(P, \Lambda)$.*

In each of the three reasoning systems (normal and extended logic programming, and default logic) the underlying concept is the choice of maximal sets of rules which have two properties: they are compatible and grounded. These sets have names specific to the reasoning system: stable models, answer sets, and extensions. The property that is of interest in this work is the notion of compatibility which we make precise, below. In the sequel the notion of groundedness will be taken in its classical sense of horn clause rules. These ideas in a graph context have been discussed [3] [4] [5] [11]. We have distinguished ourselves from these other investigations in two ways: by describing the notion of compatibility in a semantic sense. and concentrating only on the compatibility aspect in the compatibility graph [12]. In the more limited language of literals, the semantic-syntactic difference is less important, and as we will see later, the difference (only that there is a logical symbol, classication negation, which has a semantic meaning) is totally eliminated in the clique, the final computational structure.

Our purpose in this paper is to show that the basic properties of the graph and cliques are maintained in the three systems. We begin this investigation by first defining the compatibility graph. The nodes in the graph represent rules and the edges in the graph capture a relationship between rules that indicates that they are pair-wise compatible.

Given extended logic programs and default theories we define a pair-wise compatibility relation between rules.

Definition 1. *Any two rules, r_i and r_j, such that*

$$\begin{array}{l} r_i : \mathit{head}_i \leftarrow \mathit{body}_i^+, \mathit{body}_i^-, \\ r_j : \mathit{head}_j \leftarrow \mathit{body}_j^+, \mathit{body}_j^-, \end{array}$$

are 8-way pair-wise compatible *iff*

$$\begin{array}{l} \mathit{head}_i \neq \neg \mathit{head}_j, \\ \neg \mathit{head}_j \notin \mathit{body}_i^+, \\ \neg \mathit{head}_i \notin \mathit{body}_j^+, \\ (\forall b_i)(\forall b_j)(b_i \in \mathit{body}_i^+), (b_j \in \mathit{body}_j^+) \Rightarrow b_i \neq \neg b_j, \quad \mathit{and} \\ (\forall b)(\text{not}(b) \in (\mathit{body}_i^- \cup \mathit{body}_j^-) \Rightarrow b \notin (\{\mathit{head}_i\} \cup \{\mathit{head}_j\} \cup \mathit{body}_i^+ \cup \mathit{body}_j^+) \end{array}$$

The equivalent definition for the 8-way pair-wise compatibility relation between default rules in a default theory has been given previously in [12]. There the relation is given using classical logical consistency, so the relationship in that form holds for the restricted form of default logic described here, and the general case, as well.

In the settings in which the 8-way pair-wise compatibility relation will be used, the rules of classical logic are available, so, $\neg\neg x = x$. It should also be noted that the tests between the elements in *body*$^-$ and the positive parts of the rules capture the "hidden" classical negation that exists in the meaning of the rule's negative body elements.

Because there is no classical negation symbol in normal logic programs, a reduced form of the 8-way pair-wise compatibility relation is used. The reduction is the obvious projection of the 8 conditions to the language of normal logic programs.

Definition 2. *Any two rules, r_i and r_j, such that*

$$\begin{aligned} r_i &: \mathit{head}_i \leftarrow \mathit{body}_i^+, \mathit{body}_i^-, \\ r_j &: \mathit{head}_j \leftarrow \mathit{body}_j^+, \mathit{body}_j^-, \end{aligned}$$

are 4-way pair-wise compatible *iff*

$$(\forall b)(\mathrm{not}(b) \in (\mathit{body}_i^- \cup \mathit{body}_j^-) \Rightarrow b \notin (\{\mathit{head}_i\} \cup \{\mathit{head}_j\} \cup \mathit{body}_i^+ \cup \mathit{body}_j^+))$$

The rules have an obvious 'positive' part (in default logic, this corresponds to the logical elements, that is, the prerequisite and the consequence), and a 'negative' part (or non-logical part or justification in default logic), so it is useful to have terminology that combines these parts and the pair-wise compatibility relations.

Definition 3. *A rule is* positive consistent *if the positive parts of the rule are consistent. A pair of rules is* pair-wise positive consistent *if the positive parts of one rule are consistent with the positive parts of the other rule. A set of rules is* positive consistent *if every pair of rules is pair-wise positive consistent.*

The 4-way pair-wise compatibility relation can also be used to describe the relationship between the negative and positive parts of rules.

Definition 4. *A rule is* positive-negative consistent *if it is 4-way pair-wise compatible with itself. A pair of rules is* pair-wise positive-negative consistent *if the two rules are 4-way pair-wise compatible. A set of rules is* positive-negative consistent *if every pair of rules is positive-negative pair-wise compatible.*

Definition 5. *Two rules are* pair-wise compatible *if they are pair-wise positive consistent and pair-wise positive-negative consistent. A set of rules is* compatible *if every pair of rules in the set is pair-wise compatible.*

Proposition 1. *A set of compatible rules is positive consistent and positive-negative consistent.*

Proof. For sets of extended logic program rules and default rules that are 8-way-compatibility-related, the result is a simple consequence of the definition of 8-way compatibility and the fact that every pair of rules has this property. For normal program rules, any set is positive consistent [10] and positive-negative consistency results from the definition of 4-way compatibility and the fact that every pair of rules has this property.

Definition 6. *A* compatibility graph *is a graph with nodes representing rules of a logic program (resp. default theory) and edges representing a pair-wise compatibility relation.*

The foundational element for the common framework proposed here is the compatibility graph. Since we will be using this structure to find sets of compatible rules, it should now be obvious that the graph for extended logic programs and default theories has edges representing the 8-way compatibility relation, whereas the graph for normal logic programs has edges representing the 4-way compatibility relation. In the remainder of the paper the term 'compatibility graph' will be used to mean appropriately the 4-way or 8-way compatibility graph.

Proposition 2. *Cliques of a compatibility graph represent maximal sets of compatible rules.*

Proof. Cliques are maximal completely connected subgraphs. Therefore each rule is pair-wise consistent with every other rule in the clique.

Definition 7. *A logic program (resp. default theory) has the* 0-1 property *if it has at most one answer set (resp. extension).*

We can now state our fundamental proposition about cliques.

Theorem 2. *A logic program (resp. default theory) composed of compatible rules has the 0-1 property.*

Proof. Let $P = \{r_i \mid r_i = h_i \leftarrow b_i^+, b_i^-\}$ be a set of compatible rules. Let $H = \{h_i \mid h_i \leftarrow b_i\}$. By Thm 2.2(ii) in [14]: if Δ is a stable model of P, then $\Lambda = \text{not}(\mathit{NAnt}(P) - \Delta)$ is P-full and $\Delta = \mathit{Dcl}(P, \Lambda)$. Since Δ is a stable model, $\Delta \subseteq H$ ([7]). Because P is a set of compatible rules, $\mathit{NAnt}(P) \cap H = \emptyset$, therefore $\mathit{NAnt}(P) \cap \Delta = \emptyset$, for any $\Delta \subseteq H$. Therefore $\Lambda = \text{not}(\mathit{NAnt}(P))$. Since $\text{not}(\mathit{NAnt}(P))$ is unique, Λ is unique.

Since Λ contains only not-atoms, Λ^+ is empty. So, $\mathit{Dcl}(P, \Lambda)$ is the smallest set containing Λ^+ (i.e. the empty set) and is closed under rules $R(P, \Lambda)$. Since each rule has at least one not-atom in Λ, $R(P, \Lambda)$ is the unique set consisting of all of the rules of P appropriately stripped of all of the not-atoms.

The proof is analogous for extensions of default theories (Theorem 2.3 in [13]) and hence for answer sets of extended logic programs.

3 The computational pipeline

We are interested in the following common framework for computing answer sets (extensions) for logic programs (default theories).

Given a logic program (resp. default theory) P, we separate the program into two parts: the integrity constraints (those rules which have empty heads), *IC*, and $P' = P - \mathit{IC}$. We do this because the rules in *IC* do not play a logical role in the program, rather we treat them as a post-filter to filter out answer sets (resp. extensions) that are generated by the logical part, P'.

The filter, although it is shown as a single element in the pipeline, is composed of two pieces. The first component depends on the clique: for each clique, C, the members of the first component are those rules in $P' - C$. The elements of the second component are the rules in *IC*.

So that we can represent one pipeline for default logic and extended logic programming, it is important to establish that $\textbf{filter}_{DL}$, the filter for default logic, and $\textbf{filter}_{ExtLP}$, the filter for extended logic programming are isomorphic.

Theorem 3. *$\textbf{filter}_{DL}$ is isomorphic to $\textbf{filter}_{ExtLP}$*

Proof. The notion of the filter for default logic is given as Proposition 2.1 in [17]. The discussion prior to Proposition 4 in [8] shows the *modular* transformation between extended logic programs and default theories that have the same answer sets and extensions.

We are now in a position to discuss the *computational pipeline*. We first show the pipeline for extended logic programs and default theories, and the pipeline for normal logic programs.

$$\begin{tikzcd} P' \arrow[r, "\begin{array}{c}\textbf{8-way}\\ \textbf{relation}\end{array}"'] & \begin{array}{c}\scriptstyle Compatibility\\ graph\end{array} \arrow[r] & \begin{array}{c}\textit{cliques}\\ (\textit{potential GDs})\end{array} \arrow[r, "\textbf{ground}"'] & \begin{array}{c}\scriptstyle grounded\\ GDs\end{array} \arrow[r, "\textbf{filter}"'] & \begin{array}{c}\textit{ans. sets}\\ \textit{extensions}\end{array} \end{tikzcd}$$

$$\begin{tikzcd} P' \arrow[r, "\begin{array}{c}\textbf{4-way}\\ \textbf{relation}\end{array}"'] & \begin{array}{c}\scriptstyle Compatibility\\ graph\end{array} \arrow[r] & \begin{array}{c}\textit{cliques}\\ (\textit{potential HCs})\end{array} \arrow[r, "\textbf{ground}"'] & \begin{array}{c}\scriptstyle grounded\\ HCs\end{array} \arrow[r, "\textbf{filter}"'] & \begin{array}{c}\textit{stable}\\ \textit{models}\end{array} \end{tikzcd}$$

We now want to state and prove the main contribution of this paper: that the application of the computational pipelines in their respectively appropriate contexts gives the same desired outcome: the computation of stable models, answer sets, and extensions. The proof of this result does not directly compare each stage of the pipeline, rather only the outcomes are compared.

Definition 8. *Given a logic program (resp. default theory),* a set of necessary cliques *is any minimal subset of cliques from the compatibility graph that generates the set of answer sets (resp. the set of extensions).*

Lemma 1. *For each set of necessary cliques there is a bijection from it to the set of answer sets (resp. the set of extensions).*

Proof. Theorem 1 states that cliques have the 0-1 property. Since the minimality condition in the definition of necessary cliques eliminates the cliques producing no answer sets and those producing answer sets that some other clique produces, the bijection is obvious.

Theorem 4. *The cliques produced from the compatibility graph of a logic program (resp. a default theory) contains the set of necessary cliques of the logic program (resp. default theory).*

Proof. See [12] for the proof for default logic. The results of [8] show the implication for extended logic programs. The result for normal logic programs results directly from the lemma and from the fact that if the language of extended logic programs is restricted to the language of normal logic programs, the 8-way and 4-way pair-wise compatibility relations are equivalent.

4 Properties of cliques and the computational pipeline

Now that we have shown that the computational pipeline has the same effect in all three nonmonotonic reasoning systems, the computation of the stable models, answer sets, and extensions, we are in a position to state some properties of cliques in general which can now be applied to all three nonmonotonic reasoning systems. Some properties of these cliques, particular to their role in this framework, are also presented.

Property 1. There exist polynomial delay time algorithms for computing cliques of a graph. In addition, there exist polynomial delay time algorithms which compute cliques of a graph in lexicographic order if potentially exponential space is allowed. [9]

Although the time to output all cliques of a graph is bounded by polynomial total time, $(O(n^k C))$, where n is the size of the graph and C is the number of cliques, polynomial time delay guarantees that the time to produce the first and the time between producing cliques is bounded by a polynomial in n.

Property 2. Grounding each clique can be done in linear time [6].

Property 3. Filtering each potential set of ground sets of rules can be done in linear time.

No algorithm has been proposed for computing stable models (resp. answer sets, extensions of default theories), that can guarantee that the first one is produced in time bounded by a polynomial in the size of the number of rules. Of course, Property 1 concerns cliques, not stable models (resp. answer sets, extensions). The property does not consider that the logic program (resp. default theory) may have integrity constraints or that the filter may contain rules that filter potential ground sets of rules. Also, because of the 0-1 property, we could have the situation in which an exponential number of cliques are generated before one is found that has a ground subset. The first problem is rather difficult to overcome, but in some situations it is possible. But, we can recover from the second problem in a much more general way: by considering logic programs (resp. default theories) with facts (see Property 5), which in most realistic settings will probably be the case. We establish this property first, and then discuss the inital problem.

Property 4. If every set that can be composed of all of the facts, all of the horn clause rules, and one other rule of a logic program (resp. default theory) is positive consistent (this property is trivially the case for a normal logic program, since it is always positive consistent), then all of the facts will be in all cliques that survive the filter step.

Proof. If a clique is filtered for some other reason, or if the rules are positive-negative compatible, the result is obvious. If not, then the fact that is not positive-negative consistent with a rule will filter the cliques that do not have the fact, since facts are always enabled (no body (resp. no prerequisite nor justification).

We are now in a position to return to the problem of giving conditions for the extension produced by a clique not to be filtered. This will then give some properties that guarantee polynomial-delay for computing answer sets (resp. extensions).

Definition 9. *A* negative rule *has the form head* $\leftarrow$ *body*$^-$. *The analogue in default logic is the prerequisite-free rule; however, the set of prerequisite-free rules is a superset of the set of negative rules.*

Proposition 3. *A logic program (resp. default theory) composed of compatible rules has a unique answer set (resp. extension) if at least one rule is a fact or at least one rule is a negative rule (resp. prerequisite-free rule).*

So, from Proposition 3 and Property 4 we have the following:

Property 5. If P is a normal logic program (resp. extended logic program, default theory) that has at least one fact that is pair-wise consistent with all of the other rules, all cliques generated from the compatibility graph of P contain a single stable model (resp. answer set, extension).

Obviously, since cliques are logic programs composed of compatible rules, Proposition 3 applies directly to cliques. However, the 0-1 property of cliques and the filter in the computational pipeline prevents a one-to-one correspondence between cliques and stable models (resp. answer sets, extensions) in the general setting. So, results known about cliques cannot be directly translated into properties about stable models (resp. answer sets, extensions). Obviously, removing the grounding problems which creates the weakness of the 0-1 property and by having a filter in the pipeline which has does not cancel any cliques would give a one-to-one correspondence between cliques and stable models (resp. answer sets, extensions). Another problem still remains: cliques may generate duplicate stable models (resp. answer sets, extensions). We have, however, a natural class of default theories which do not have any of these problems.

Lemma 2. *A prerequisite-free normal default theory without integrity constraints (resp. an equivalent kind of extended logic program) has a filter step which does no filtering.*

Proof. Each filtering rule must be pair-wise incompatible with some rule in the clique. Therefore, the incompatibility is guaranteed to be with the justification. Thus, since each rule in the clique represents a fact (see the proof of Proposition 4), no rule in the filter is enabled by having its justification compatible with the unique extension (resp. answer set) generated by the clique.

Proposition 4. *For the compatibility graph of a consistent prerequisite-free normal default theory (with no duplicate rules), (resp. an equivalent extended logic program) each clique has a unique extension.*

Proof. Each clique represents the Horn theory in which each Horn rule corresponds to the default rules with the justifications of each rule removed. Since the rules are prerequisite-free, the Horn rules are facts. Each clique is unique, so each set of facts is unique.

Corollary 1. *Extensions of a prerequisite-free normal default theory with no integrity constraints (resp. answer sets of the equivalent kind of extended logic program) can be produced with polynomial-delay.*

We can also eliminate the filter step by considering other kinds of extensions.

Corollary 2. *The extension possessed by any clique from the compatibility graph of a default theory P which contains at least one fact and no integrity constraints is a Lukasiewicz-extension.*

Corollary 3. *The first Lukasiewicz-extension of a default theory which contains at least one fact that is pair-wise consistent with all of the other rules and no integrity constraints can be produced in time bounded by a polynomial.*

With the current version of the clique enumeration tree, an extension can occur in multiple cliques. Once we have results regarding the pruning of the clique enumeration tree (see Section 5), we should be able to give a stronger result for all Lukaseiwicz-extensions (perhaps polynomial-delay).

5 Future directions

Although the framework based on cliques, presented in the manner above, has proven useful to study analogous properties of the three nonmonotonic reasoning systems, it is enticing to ask whether the framework can be used computationally. The obvious gain is that the same computational framework can be used for all three reasoning systems.

Of course, a direct implementation of the framework, as presented, is inappropriate for computation. Algorithms to enumerate cliques (without duplicates) generate the cliques as leaves in a tree. As shown, grounding and filtering is only done on the leaves of this tree, after the complete clique enumeration tree is produced. This naïve approach generates $O(2^n)$ cliques, requiring exponential space in worst case to store the leaves of the clique enumeration tree. Obviously, the grounding and filtering should be interleaved with the clique generation. But even with this change, computational problems still remain. Grounding can produce the same result from different cliques. To delete duplicates, exponential space may be required. Filtering can reduce different potential sets based on a common pattern, but by only looking at the leaves of the tree, the search may thrash needlessly.

The future work that we sketch in the following paragraphs is one method to move the filtering and grounding decisions to internal nodes of the clique enumeration tree with the guarantee of keeping all stable models (resp. answer sets, extensions). The details of this work will be reported elsewhere.

Each path in the clique enumeration tree represents compatible rules. The only decisions that still remain concern rule grounding alone. This is obviously true for the grounding element in the pipeline. It is also the case for the integrity constraint filtering decisions, since if an integrity constraint is grounded by the grounded clique rules, it filters the clique. The non-integrity constraint filter rules require full information about their filtering status. These rules are dealt with differently.

What is done with this grounding information differs: In the first case having knowledge that a rule cannot be grounded may allow rules that appear later in the clique enumeration to be temporarily removed from the compatibility graph. To have knowledge that an integrity constraint is grounded allows this filtering rule to prune the clique enumeration. As soon as an integrity constraint is grounded, the subtree rooted by the node that grounds the integrity constraint can be pruned and a backtrack in the clique enumeration tree can occur. Non-integrity constraint filter rules that cannot be grounded can be removed from the filter.

This discussion can be made a bit more precise by showing a method to select the nodes in the clique enumeration tree at which these grounding decisions can be made using a modified form of relaxed stratification [2], a computationally inexpensive binary relationship between rules. Rules are related if an atom in the consequence of one rule is contained in the other rule. A stratification of the rules of the program can be created and the strata ordered such that the rules in a strata are not related to rules in prior strata.

The modifications to relaxed stratification are quite simple: it is defined initially for the program as a whole, but it is dynamic, so as the enumeration progresses, the relaxed stratification can be modified (as rules are pruned, the strata and ordering can be refined); since the compatibility relation has already removed any computational effects of the *body*$^-$, the relationship between two rules needs to take into account only the head of one rule and the *body*$^+$ (resp. prerequisites) of the other rule; and the strata are ordered by the criterion: rules are not related to rules in *following* strata.

The strata given by relaxed stratification can be used as points for deciding whether it is impossible to ground a rule. Any rules that are still to be considered on the current path by the clique enumeration algorithm that are determined to be ungroundable can be temporarily removed from the compatibility graph. They are returned to the graph upon a backtrack in the clique enumeration tree. Filter rules can be temporarily removed from the filter and returned to the filter upon a backtrack, if they are deemed ungroundable in the current clique enumeration path. The other procedure that is undertaken at the stratum boundary is to remove duplicate partial solutions. (The space requirements are $O(2^s)$, where s is the number of rules in the stratum.)

When leaves in the clique enumeration tree are reached, any remaining integrity constraints in the filter are removed. The remaining filter rules are checked to see if any are enabled by (are both grounded and consistent) with the stable model (resp. answer set, extension) candidate. If so, the candidate is filtered. Any remaining stable models (resp. answer sets, extensions) are guaranteed to be unique.

In worst case the computational problems are not removed by these heuristics. So, a study of the computational behaviour over the standard benchmarks is obviously necessary.

If the method sketched above proves to be reasonable, it is a method that can be used for all three nonomonotonic reasoning systems. That cliques contain compatible rules in all three reasoning systems indicates that classical negation plays no logical role in cliques (so at the level of clique, the inclusion of classical negation in the language has no theoretical impact). The compatibility graph has removed the semantic effect of classical negation and has given it a purely syntactic role. In other words, we have a preprocessing step that allows the rewriting of extended logic programs (resp. default theories) as normal logic programs but maintaining the overall semantic effect of negation. With this in mind, eliminating the expressivity of classical negation in the language seems to be a disadvantage that may be unnecessary. Reintroducing classical negation would allow the recapturing of normal and semi-normal defaults, and would deliver all of the natural expressivity of classical negation.

6 Related work

Other work has focused on the grounding and independence properties in graphs (kernels of graphs, 0-1 graphs, block-support graphs) [3] [4] [5] [11]. This means that the independence (anti-relation to compatibility) and dominance (grounding) in the graphs representing the logic programs (resp. default theories) are kept together. We have separated these two aspects of the problem and shown some advantages for doing this.

Splitting default theories has been proposed by [2] and [18], and [1] has compared the two methods. Cliques was originally proposed as a method for splitting default theories in [12], and with the results herein, has been shown to be applicable to extended logic programs and normal logic programs.

7 Acknowledgements

A number of people have helped push these ideas. This research has been partially supported by the National Science and Engineering Research Council (Canada) through Research Grant 0036853.

References

1. Grigoris Antoniou. A comparison of two approaches to splitting default theories. In *AAAI/IAAI*, pages 424–429, 1997.
2. P. Cholewiński, V. W. Marek, M. Truszczyński, and A. Mikitiuk. Computing with default logic. *Artificial Intelligence*, 112:105–146, 1999.
3. Y. Dimopoulos and V. Magirou. A graph-theoretic approach to default logic. *Information and Computation*, 112:239–256, 1994.
4. Y. Dimopoulos, V. Magirou, and C. H. Papadimitriou. On kernels, defaults and even graphs. *Annals of Mathematics and Artificial Intelligence*, 20:1–12, 1997.
5. Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science*, 170:209–244, 1996.
6. W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisability of propositional horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.
7. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, 1988.

8. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
9. P. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.
10. V. Lifschitz. Foundations of logic programming. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 69–127. The University of Chicago Press, 1996.
11. T. Linke, C. Anger, and K. Konczak. More on `nomore`. In S. Flesca, S. Greco, N. Leone, and G.Ianni, editors, *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02), Lecture Notes in Artificial Intelligence*, volume 2424, pages 468–480, 2002.
12. R. E. Mercer, L. Forget, and V. Risch. Comparing a pair-wise compatibility heuristic and relaxed stratification: Some preliminary results. In S. Benferhat and Ph. Besnard, editors, *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 2143, pages 580–591. Springer-Verlag, 2001.
13. Ilkka Niemelä. Towards efficient default reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 312–318, Montreal, Canada, August 1995. Morgan Kaufmann Publishers.
14. Ilkka Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. Fachbericht Informatik 7–96, Universität Koblenz-Landau, 1996. Available at http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/.
15. Ilkka Niemelä and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 420–429, Dagstuhl, Germany, July 1997. Springer-Verlag. I have not seen a copy of this paper. It is not online at www.springer.de.
16. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
17. V. Risch and C. Schwind. Tableau-based characterization and theorem proving for default logic. *Journal of Automated Reasoning*, 13:223–242, 1994.
18. H. Turner. Splitting a default theory. In *Proceedings of AAAI-96*, pages 645–651, 1996.