Proceedings of the 12th International Conference on Logic Programming, MIT Press, Cambridge, MA., 1995

Experimenting with Nonmonotonic Reasoning

Paweł Cholewiński, V. Wiktor Marek,
Artur Mikitiuk, Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046
`{pawel|marek|artur|mirek}@cs.engr.uky.edu`

Abstract

In this paper, we describe a system, called TheoryBase, whose goal is to facilitate experimental studies of nonmonotonic reasoning systems. TheoryBase generates test default theories and logic programs. It has an identification system for generated theories, which allows us to reconstruct a logic program or a default theory from its identifier. Hence, exchanging test cases requires only exchanging identifiers. TheoryBase can generate a large variety of examples of default theories and logic programs. We believe that its universal adoption may significantly advance experimental studies of nonmonotonic reasoning systems.

1 Introduction

Nonmonotonic reasoning has been introduced in the 1970s [21, 31]. The first full-fledged nonmonotonic formalisms — circumscription, default logic and modal nonmonotonic logics — were proposed in early 1980s [20, 32, 23, 22, 24, 25]. Initially, understanding commonsense reasoning and knowledge representation applications served as main motivations driving the development of the discipline. In the late 1980s, however, it was observed that nonmonotonic logics offer insights on semantics for negation in logic programming [9, 17, 3]. Since then, the connections between logic programming and nonmonotonic reasoning have been extensively studied [30, 1]. These efforts related several semantics for logic programs (supported semantics, stable semantics, well-founded semantics) to objects (expansions or extensions) one assigns to theories in nonmonotonic reasoning. For an overview of nonmonotonic systems as well as notation the reader is referred to [19].

Despite advances in our understanding of nonmonotonic logics, implementation efforts and experimentation with nonmonotonic reasoning systems have been lagging behind. There are only few reported examples of such work [29, 10, 7, 35, 2]. If the area of nonmonotonic reasoning is to grow, this state of affairs must change. It is especially true in view of several recent complexity results [12, 34, 27]. This work shows that, in contrast with early hopes [21], nonmonotonic reasoning is not simpler than classical logic reasoning. Basic decision problems underlying nonmonotonic reasoning are complete for classes at the second level of the polynomial hierarchy. Even significantly restricted versions of these problems remain NP- or coNP-complete [18, 14]. Is there, then, any hope that nonmonotonic reasoning will prove practical? Can it be made competitive with automated reasoning based on classical logic?

Complexity results tell only half of the story. Nonmonotonic theories encoding graph problems are often smaller in size than corresponding classical logic descriptions. Hence, loss in complexity may be offset by the drop in the size of input (see [5]). Systematic experimentation may be the most direct way and, perhaps, the only way to answer the above questions. In addition to complexity results mentioned earlier, several algorithms for reasoning from default theories and logic programs with negation were presented in [19, 13, 28, 26]. In view of the large body of these results, it seems that the main problem hindering the research on these algorithms and the efforts to develop reasoning systems based on nonmonotonic logics is the lack of adequate experimentation testbed. In this paper we address this issue.

To experiment with software systems we need easily generated, realistic and meaningful input instances. The input data should be easy to reproduce and disseminate so that other researchers can use it. The generation method should allow the users to control basic parameters of test cases to facilitate comprehensive testing of the software.

The problem of producing sets of benchmark data appears in all areas of experimental research. A possible approach is to produce and distribute a database of real-life examples. For example, databases of benchmarks were created for experimentation with linear programming algorithms, for investigation of methods to solve the travelling salesman problem, for a number of VLSI problems, and in numerous other areas of experimental research in computer science. The benefits of this method are evident. The problems are realistic and meaningful, and they can easily be disseminated. But there are also drawbacks. The data can be used only in a specific application domain and often does not provide enough flexibility to allow full-fledged testing. The other approach frequently used in experimental research is to generate data randomly. This approach offers an unlimited number of test cases and the user has control over at least some important parameters of data generated. For example, when generating random graphs, one can request a specific number of vertices and edges. However, the data generated randomly has often properties that rarely occur in real-life examples. It is well known that (under some technical assumptions) almost every connected random graph is hamiltonian [4]. Similarly, it is now believed that random 3-SAT problems do not provide an adequate model for problems likely to occur in real-life applications [11, 6].

None of these approaches has been fully developed for experimenting with logic programming or nonmonotonic reasoning. In logic programming research, benchmark programs usually come from a small set of problems including the "naive reverse" program [33]. Moreover, no notion of a random logic program or default theory has been proposed yet.

The proposal we are describing in this paper is based on the work by Knuth [15] on methods to generate graphs, and on our results providing encodings of graph problems in terms of default theories and logic programs.

Knuth in [15] argues that random graphs do not constitute an adequate tool for testing graph algorithms. Instead, Knuth develops a graph generation system, the Stanford GraphBase. This system is publicly available (see [15] for details) and, thus, can be used as a "common denominator" for work requiring experimenting with graphs. The core of the system is formed by several graph generating procedures. Some of the methods employed root the graphs they generate in real-life objects such as maps and dictionaries. At the same time, these methods are flexible and can generate large families of graphs preserving some randomness appearance. An important feature of GraphBase is that every graph generated gets a unique label (or identifier). It is essential for storing and easy reproduction of test cases generated.

The main contribution of this paper is an extension of the Stanford GraphBase to a system that generates logic programs and default theories. Our idea is simple. Problems on graphs such as coloring, hamiltonicity, existence of kernels, have encodings as default theories and logic programs. This is, of course, implied by complexity considerations [18, 12, 16]. However, to build a system, we need *explicit* encodings. Several such encodings will be presented in Section 2 of this paper.

To generate test theories, we propose to first generate graphs (using GraphBase) and then to encode problems (coloring, hamiltonicity, etc.) for these graphs as logic programs and default theories. By coupling Knuth's GraphBase and these translations we get a system for creating examples of *meaningful*, interesting, and yet sufficiently randomized default theories and logic programs. This system will be referred to as TheoryBase.

The identification system for TheoryBase allows us to easily reproduce theories out of their identifiers. Thus, it facilitates an exchange of test default theories and logic programs. In this fashion, a useful and functional system of benchmarks for nonmonotonic reasoning can be created.

TheoryBase was motivated by Default Reasoning System project, DeReS, currently being carried out at the University of Kentucky. The need to build an experimentation testbed for DeReS prompted our TheoryBase project. Several elements of DeReS are now completed (propositional logic programming with stable semantics, default logic with Reiter's extensions) and we are now beginning a systematic experimentation effort.

The paper consists of two main parts. The next section presents encodings of graph problems as logic programs and default theories. Section 3 describes how these encodings are used in the design of TheoryBase.

2 Default theories and logic programs for graph problems

In [18] it was shown that the problem of existence of stable models of a logic program is NP-complete. This result implies that for every problem $\mathcal{P}$ in NP there is a polynomially constructible encoding of $\mathcal{P}$ as a logic program, say $lp(\mathcal{P})$, such that $\mathcal{P}$ has a solution if and only if the program $lp(\mathcal{P})$ has a stable model. Similarly, the Σ_2^P-completeness of existence of extensions for a default theory [12] implies that there are efficient encodings of problems in the class Σ_2^P as default theories. These encodings form the core of our testbed system — TheoryBase. The problems we focus on in this paper are: maximal independent sets and maximal matchings, colorings, existence of kernels and existence of hamiltonian cycles (see [8] for graph theory terminology). For each of these problems we will present one or more encodings in nonmonotonic formalisms together with theorems asserting the correspondence between the graph problem and the encoding. These problems appear in numerous applications. Hence, as long as underlying graphs are realistic, our logic programs and default theories have meaningful interpretations.

The complexity results show only existence of encodings. To build TheoryBase, we need their explicit description. This is the topic of this section.

In the paper we are using the following notation. By $G = (V, E)$ we denote a graph with the vertex set V and the edge set E. The set E consists of two-element sets or ordered pairs of vertices, depending on whether the graph is undirected or directed. We also denote $|V|$ by n and $|E|$ by m. For undirected graphs we denote the set of neighbors of a vertex v by $\Gamma(v)$. For directed graphs we define

$$\Gamma^+(v) = \{w \in V \colon (v,w) \in E \ \text{ and } \ v \neq w\}$$

and

$$\Gamma^-(v) = \{w \in V \colon (w,v) \in E \ \text{ and } \ v \neq w\}.$$

2.1 Maximal independent sets and maximal matchings

Let $G = (V, E)$ be an undirected graph. For every $v \in V$, we define a default

$$d_v = \frac{: \neg in(v_1), \ldots, \neg in(v_k)}{in(v)},$$

where $\Gamma(v) = \{v_1, \ldots, v_k\}$ and $in(v_i)$ are propositional atoms (intuitively, $in(v)$ says that v is *in* an independent set).

Theorem 2.1 *Let $G = (V, E)$ be an undirected graph and let $D = \{d_v : v \in V\}$. A set $U \subseteq V$ is a maximal independent set in G if and only if $Cn(\{in(v) \colon v \in U\})$ is an extension for the default theory $\Delta_{ind} = (D, \emptyset)$.* $\square$

As a corollary, we obtain an encoding for the maximal matching problem. For every $e \in E$, define a default

$$d_e = \frac{:\neg in(e_1), \ldots, \neg in(e_k)}{in(e)},$$

where $e_1, \ldots, e_k$ are all the edges adjacent to e and $in(e_i)$ are propositional atoms.

Theorem 2.2 *Let $G = (V, E)$ be an undirected graph and let $D = \{d_e : e \in E\}$. A set $F \subseteq E$ is a maximal matching in G if and only if $Cn(\{in(e){:}\, e \in F\})$ is an extension for the default theory $\Delta_{match} = (D, \emptyset)$.* □

Remark 2.1 The encoding of the maximal independent set problem uses n defaults and the encoding of the maximal matching problem uses m defaults. The size of the first encoding is $O(2m+n)$. The size of the second encoding is $O(mK)$, where K is the largest vertex degree in G.

Remark 2.2 Both problems presented above can be encoded as logic programs. For example, using a translation described in [17], each default d_v can be expressed by the clause

$$in(v) \leftarrow \mathbf{not}(in(v_1)), \ldots, \mathbf{not}(in(v_k)).$$

Under this encoding, maximal independent sets correspond to stable models of the resulting logic program.

2.2 Colorings

Let $G = (V, E)$ be an undirected graph with the set of vertices $\{v_1, \ldots, v_n\}$, where each v_i is a positive integer. Let $C = \{c_1, \ldots, c_k\}$ be a set of colors. A coloring of G is any mapping $f : V \to C$. Given a coloring f, $R(f)$ denotes the set of propositional atoms $\{clr(v, f(v)) : v \in V\}$. This set represents an assignment of colors to the vertices of G.

For each vertex $v_i, i = 1, \ldots, n$, and for each color $c_j, j = 1, \ldots, k$, we define the default rule

$$color(v_i, c_j) = \frac{:\neg clr(v_i, c_1), ..., \neg clr(v_i, c_{j-1}), \neg clr(v_i, c_{j+1}), ..., \neg clr(v_i, c_k)}{clr(v_i, c_j)}.$$

The set of default rules $\{color(v_i, c_j) : j = 1, \ldots, k\}$ is used to assign exactly one color to a vertex v_i. Let

$$D_0 = \{color(v_i, c_j) : i = 1, \ldots, n, \quad j = 1, \ldots, k\}.$$

The default theory $(D_0, \emptyset)$ has k^n extensions corresponding to all possible colorings (not necessarily proper) of the vertices of G. To describe *proper colorings*, that is, colorings where any two vertices connected by an edge have different colors, we will use additional default rules. These rules will *kill* extensions which define "non-proper" colorings. They will be called *killing* defaults. To describe them we define several auxiliary formulas.

Let $E = \{e_1, \ldots, e_m\}$ and let $e_l = \{x_l, y_l\}$, where $x_l, y_l \in V$ are the endpoints of e_l. By $\varphi(l,j)$ we denote the following formula:

$$\varphi(l,j) = clr(x_l, c_j) \wedge clr(y_l, c_j).$$

The formula $\psi(l) = \bigvee_{j=1}^{k} \varphi(l,j)$ expresses the fact that both end vertices of the edge e_l have the same color. For a vertex v_i, let E_i be the set of all edges connecting v_i to smaller vertices. That is,

$$E_i = \{e \in E : e = \{v_i, w\} \quad \text{and} \quad w < v_i\}.$$

(The condition $w < v_i$ ensures that every edge appears in exactly one set E_i.) Now, for every vertex $v_i \in V$ define

$$\Phi_i = \bigvee_{e_l \in E_i} \psi(l)$$

(if $E_i = \emptyset$ then we assume that $\Phi_i = \bot$). The formulas Φ_i are used to ensure that no two connected vertices receive the same color. Finally, let F, F_i and $F_{l,j}$ be arbitrary auxiliary atoms, that is any atoms different from the atoms of the form $clr(v,c)$. The intuitive meaning of these auxiliary atoms is *falsity*. They are used to prevent undesirable theories from being generated as solutions. We define the following *killing* default rules:

$$global = \frac{\bigvee_{l=1}^{m} \bigvee_{j=1}^{k} \varphi(l,j) : \neg F}{F},$$

$$partial(i) = \frac{\Phi_i : \neg F_i}{F_i}, \quad i = 1, \ldots, n,$$

$$local(l,j) = \frac{\varphi(l,j) : \neg F_{l,j}}{F_{l,j}}, \quad l = 1, \ldots, m, \quad j = 1, \ldots, k.$$

Default *global* performs a global (over all edges) check of the proper coloring condition. Default $partial(i)$ verifies the condition for all edges in E_i. Finally, default $local(l,j)$ verifies the proper coloring condition for edge e_l and color j.

Theorem 2.3 *Let $G = (V, E)$ be an undirected graph. For any of the following default theories:*

1. $\Delta^1_{col} = (D_0 \cup \{global\}, \emptyset)$,

2. $\Delta^2_{col} = (D_0 \cup \{partial(i) : i = 1, \ldots, n\}, \emptyset)$,

3. $\Delta^3_{col} = (D_0 \cup \{local(l,j) : l = 1, \ldots, m, j = 1, \ldots, k\}, \emptyset)$,

if $f : V \to C$ is a proper coloring of G then $Cn(R(f))$ is an extension for Δ^i_{col}, $i = 1, 2, 3$. Moreover, if S is an extension for Δ^i_{col}, for some $i = 1, 2, 3$, then $S = Cn(R(f))$ for some proper coloring f of G. $\square$

Remark 2.3 We presented three encodings of the graph coloring problem. They differ in the technique used to check that a coloring is proper. All these theories have nk default rules to generate a coloring and some additional default rules to ensure that the coloring is proper. The total number of default rules is $nk + 1$ for Δ^1_{col}, $nk + n$ for Δ^2_{col} and $nk + mk$ for Δ^3_{col}. However, the length of the resulting default theory is approximately the same in all cases and is given by $O((nk + m)k)$.

Remark 2.4 The encoding Δ^3_{col} has a straightforward translation into a logic program. To this end just replace each default rule $color(v_i, c_j)$ by a logic program clause

$$\begin{aligned} clr(v_i, c_j) \quad \leftarrow \quad & \mathbf{not}(clr(v_i, c_1)), \ldots, \mathbf{not}(clr(v_i, c_{j-1})), \mathbf{not}(clr(v_i, c_{j+1})), \\ & \ldots, \mathbf{not}(clr(v_i, c_k)) \end{aligned}$$

and each default $local(l, j)$ by a logic program clause

$$F_{l,j} \leftarrow clr(x_l, c_j), clr(y_l, c_j), \mathbf{not}(F_{l,j}).$$

2.3 Kernels

Let $G = (V, E)$ be a directed graph with the set of vertices $\{v_1, \ldots, v_n\}$. Let us identify the set of vertices of G with a set of atoms of a propositional language. Let P_G be the following logic program:

$$P_G = \{v \leftarrow \mathbf{not}(w) : (v, w) \in E\}.$$

It was shown [18] that the stable models of P_G are precisely the sets of the form $V \setminus K$ where K is a kernel of G. The straightforward rewriting of the clause $c = v \leftarrow \mathbf{not}(w)$ as a default rule $d_c = \frac{:\neg w}{v}$ leads to a default theory whose extensions correspond to kernels of G. Specifically, K is a kernel of G if and only if $Cn(V \setminus K)$ is an extension of the default theory $\Delta^1_{ker} = (\{d_c : c \in P_G\}, \emptyset)$.

Here we present two other encodings of kernels in the language of default logic. For each vertex $v \in V$ we will use a propositional atom $out(v)$ with the intuitive meaning that v is *out* of kernel. First, we define two default rules for each vertex $v_i \in V$:

$$\frac{: out(v_i)}{out(v_i)}, \qquad \frac{: \neg out(v_i)}{\neg out(v_i)}. \tag{1}$$

Let

$$D_0 = \bigcup_{i=1}^{n} \left\{ \frac{: out(v_i)}{out(v_i)}, \frac{: \neg out(v_i)}{\neg out(v_i)} \right\}. \tag{2}$$

The default theory $(D_0, \emptyset)$ has 2^n extensions corresponding to all subsets of V. To leave only those subsets of V whose complements correspond to kernels of G, we need to use either additional propositional formulas or additional defaults. We will explore both possibilities.

To ensure that vertices of the kernel form an independent set we will use propositional formulas

$$\varphi(v, w) = out(v) \vee out(w),$$

where $(v, w) \in E$. To ensure that for a vertex v which does not belong to the kernel there is an edge $(v, w) \in E$ such that w is in the kernel we will use the formula

$$\psi(v) = out(v) \supset \neg out(w_1) \vee \ldots \vee \neg out(w_i), \quad \text{where} \quad \Gamma^+(v) = \{w_1, \ldots, w_i\}.$$

Now, we can state our first result on kernels.

Theorem 2.4 *Let $G = (V, E)$ be a directed graph and let $\Delta^2_{ker} = (D_0, W)$ be a default theory such that*

$$W = \{\varphi(v, w) : (v, w) \in E\} \cup \{\psi(v) : v \in V\}.$$

If $K \subseteq V$ is a kernel of G then $Cn(\{\neg out(v) : v \in K\} \cup \{out(v) : v \notin K\})$ is a consistent extension for Δ^2_{ker}. Conversely, if S is a consistent extension for Δ^2_{ker} then S is of the form $Cn(\{\neg out(v) : v \in K\} \cup \{out(v) : v \notin K\})$, where $K \subseteq V$ and K is a kernel of G. □

The third encoding of the kernel problem will be obtained by replacing propositional formulas from W in Theorem 2.4 by default rules. To this end we define the following *killing* defaults:

$$ind(v, w) = \frac{\neg out(v) \wedge \neg out(w) : \neg F_{v,w}}{F_{v,w}}, \quad (v, w) \in E,$$

$$dom(v) = \frac{out(v) : out(w_1), \ldots, out(w_i), \neg F_v}{F_v}, \quad v \in V,$$

where, as previously, $\Gamma^+(v) = \{w_1, \ldots, w_i\}$ and $F_{v,w}$'s and F_v's are arbitrary auxiliary atoms different from the atoms of the form $out(v)$. Defaults $\{ind(v, w) : (v, w) \in E\}$ ensure that the set of selected vertices is an independent set. Defaults $\{dom(v) : v \in V\}$ ensure that the set of selected vertices is a dominating set.

Theorem 2.5 *Let $G = (V, E)$ be a directed graph and let $\Delta^3_{ker} = (D_0 \cup D_1, \emptyset)$ be a default theory where D_0 is defined by (2) and*

$$D_1 = \{ind(v, w) : (v, w) \in E\} \cup \{dom(v) : v \in V\}.$$

If $K \subseteq V$ is a kernel of G then $Cn(\{\neg out(v) : v \in K\} \cup \{out(v) : v \notin K\})$ is an extension for Δ^3_{ker}. Conversely, if S is an extension for Δ^3_{ker} then S is of the form $Cn(\{\neg out(v) : v \in K\} \cup \{out(v) : v \notin K\})$ where $K \subseteq V$ and K is a kernel of G. □

Remark 2.5 The encodings Δ^2_{ker} and Δ^3_{ker} differ in the way they verify kernel conditions. In both cases we use $2n$ default rules to generate a subset of V. In the case of Δ^2_{ker} we use, in addition, $n+m$ propositional formulas to check whether a given subset is a kernel. In the case of Δ^3_{ker} we use $n+m$ additional default rules to verify conditions for a kernel. Thus, the total number of default rules is $2n$ for Δ^2_{ker} and $3n+m$ for Δ^3_{ker}. But in Δ^2_{ker} we have $n+m$ propositional formulas in W, while in the case of Δ^3_{ker} the set W is empty. Therefore, in both cases the total length of encoding is proportional to $3n+m$, that is, it is bounded by $O(n+m)$.

Remark 2.6 The encoding Δ^3_{ker} can be translated into a logic program. To this end we will use a positive literal $in(v)$ instead of a negative literal $\neg out(v)$. Now, a pair of default rules (1) can be rewritten as a pair of logic program clauses

$$out(v_i) \leftarrow \mathbf{not}(in(v_i)) \qquad \text{and} \qquad in(v_i) \leftarrow \mathbf{not}(out(v_i)).$$

The default rule $ind(v,w)$ can be replaced with a logic program clause

$$F_{v,w} \leftarrow \mathbf{not}(out(v)), \mathbf{not}(out(w)), \mathbf{not}(F_{v,w})$$

and the default rule $dom(v)$ with a clause

$$F_v \leftarrow out(v), out(w_1), \ldots, out(w_i), \mathbf{not}(F_v).$$

2.4 Directed hamiltonian cycles

Let $G=(V,E)$ be a directed graph with the set of vertices $\{v_1,\ldots,v_n\}$. We assume that $n \geq 3$. We will construct a default theory (D,W) whose extensions correspond to hamiltonian cycles in G. We will use atoms $vstd(v)$ (intuitively, v has been visited) and $e(v,w)$ (intuitively, (v,w) is an edge of a hamiltonian cycle).

Let $W=\{vstd(v_1)\}$. Let $v \in V$ and $\Gamma^+(v)=\{w_1,\ldots,w_k\}$. We define k default rules for vertex v in the following way:

$$move(v,w_j) = \frac{vstd(v) : \neg e(v,w_1), \ldots, \neg e(v,w_{j-1}), \neg e(v,w_{j+1}), \ldots, \neg e(v,w_k)}{e(v,w_j) \wedge vstd(w_j)}.$$

Now we define the first group of defaults:

$$D_0 = \{move(v,w) : v \in V \text{ and } w \in \Gamma^+(v)\}. \qquad (3)$$

The role of the defaults in D_0 is to select one outcoming edge for every *"visited"* vertex. To make sure that every vertex is visited, we define the second group of default rules using additional new atoms $F_v, v \in V$:

$$D_1 = \left\{ \frac{: \neg vstd(v), \neg F_v}{F_v} : v \in V \right\}. \qquad (4)$$

Finally, to make sure that the edges selected by the default rules from D_0 form a cycle, we need an additional default:

$$cycle = \frac{: \neg e(u_1, v_1), \ldots, \neg e(u_k, v_1), \neg F}{F},$$

where $\Gamma^-(v_1) = \{u_1, \ldots, u_k\}$ and F is an auxiliary atom. We have the following theorem.

Theorem 2.6 *Let $G = (V, E)$ be a directed graph. Let $W = \{vstd(v_1)\}$ ($v_1 \in V$) and $\Delta_{ham} = (D_0 \cup D_1 \cup \{cycle\}, W)$, where D_0 and D_1 are defined by (3) and (4). If S is an extension for Δ_{ham} then the set of edges $C = \{(v, w) : S \models e(v, w)\}$ is a hamiltonian cycle in G. Also, if C is a hamiltonian cycle in G then*

$$Cn(\{e(v, w) : (v, w) \in C\} \cup \{vstd(v) : v \in V\})$$

is an extension for Δ_{ham}. □

Remark 2.7 The encoding of hamiltonian cycles presented in this section uses at most $m + n + 1$ default rules. The length of the resulting default theory is $O(mK + n)$, where K is the maximal vertex outdegree in G. The theory Δ_{ham} presented in this section can be rewritten as a logic program using the translation defined in [17].

3 TheoryBase

In this section we will briefly describe TheoryBase. TheoryBase is an extension of the Stanford GraphBase [15], created by Donald E. Knuth at the Stanford University. The Stanford GraphBase is a collection of datasets, procedures which generate graphs and several demo programs. It allows the users to generate families of directed, undirected, weighted, unweighted, bipartite, planar, regular and random graphs. The Stanford GraphBase also contains procedures to generate graphs by means of graph operations such as union, intersection, complement, cartesian, direct or strong product. Other graph operations are also available. The Stanford GraphBase procedures make it possible to generate permutation graphs and partition graphs. An interesting family of graphs can be generated from a table of highway distances between 128 North American cities. For detailed description of the Stanford GraphBase see [15]. Each graph generated from GraphBase has an identifier from which the graph can be reconstructed.

TheoryBase is a tool to create default theories and logic programs for experimentation with nonmonotonic reasoning systems. The main idea is to apply the encodings presented in Section 2 to graphs which are the outputs of GraphBase.

TheoryBase associates an identifier with each theory it generates. This is an extension of the concept of a graph identifier in GraphBase. A TheoryBase identifier is a pair (α, β), where α is a GraphBase identifier and

β is the name of a translation described in Section 2. We described here nine translations into default theories: Δ_{ind}, Δ_{match}, Δ^1_{col}, Δ^2_{col}, Δ^3_{col}, Δ^1_{ker}, Δ^2_{ker}, Δ^3_{ker} and Δ_{ham} (several modifications of these nine translations are also available). In addition, as we pointed out in Section 2, some of them (Δ_{ind}, Δ_{match}, Δ^3_{col}, Δ^1_{ker}, Δ^3_{ker} and Δ_{ham}) imply encodings into logic programs.

The semantics of a TheoryBase identifier (α, β) is given by a logic program or a default theory obtained by the following process:

1. generate a graph G from the GraphBase identifier α (α encodes the generation method to be used),

2. produce a logic program or a default theory by applying the translation β to G.

It should be clear that an identifier uniquely determines a logic program or a default theory. Hence, to exchange test cases, it is sufficient to exchange their TheoryBase identifiers only.

We will present now two graphs from GraphBase and two corresponding theories from TheoryBase.

The first theory is determined by the TheoryBase identifier (α, β), where:

$\alpha =$ complement(gunion(board(7,0,0,0,2,0,0), simplex(2,2,2,2,0,0,0),0,0),0,0,0),
$\beta = \Delta_{ind}$.

Its first component is a GraphBase identifier of the graph G_1 shown in Figure 1. The second part, Δ_{ind}, indicates the encoding used to produce a theory (in this case, the encoding describing maximal independent sets). The resulting set of defaults is given below:

$$\frac{:\neg in(2),\neg in(4),\neg in(5),\neg in(6)}{in(0)} \quad \frac{:\neg in(5),\neg in(6)}{in(1)} \quad \frac{:\neg in(0),\neg in(3),\neg in(5),\neg in(6)}{in(2)}$$

$$\frac{:\neg in(2),\neg in(6)}{in(3)} \quad \frac{:\neg in(0),\neg in(6)}{in(4)} \quad \frac{:\neg in(0),\neg in(1),\neg in(2),\neg in(6)}{in(5)}$$

$$\frac{:\neg in(0),\neg in(1),\neg in(2),\neg in(3),\neg in(4),\neg in(5)}{in(6)}.$$

It is easy to see that $Cn(\{in(0), in(1), in(3)\})$ is an extension of this default theory. It corresponds to the maximal independent set $\{0, 1, 3\}$ in G_1.

Next, we show the theory determined by the identifier

$$id = (\text{econ}(8, 0, 10000, 0), \Delta_{ham}).$$

The graph G_2 identified by the GraphBase identifier econ$(8, 0, 10000, 0)$ is also shown in Figure 1.

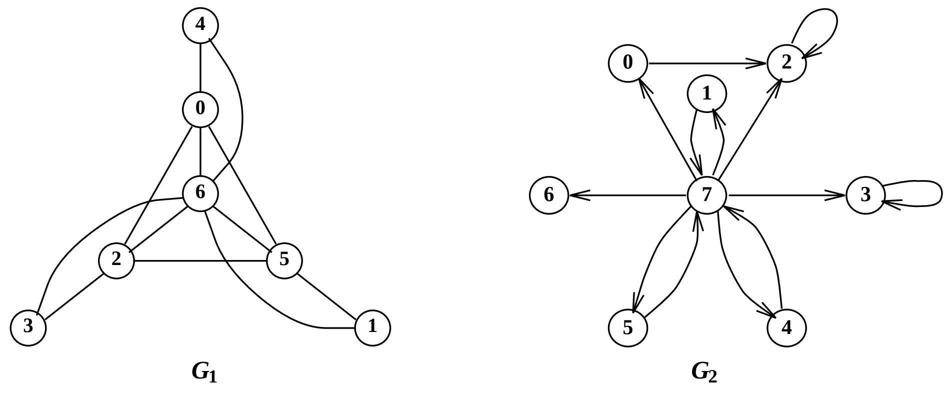

Figure 1: Graphs G_1 and G_2

The second component of the identifier id is Δ_{ham}. Hence, the theory described by id encodes the problem of existence of hamiltonian cycles for G_2. Its defaults are listed below.

$$\frac{vstd(0):}{e(0,2)\wedge vstd(2)} \qquad \frac{vstd(1):}{e(1,7)\wedge vstd(7)} \qquad \frac{vstd(4):}{e(4,7)\wedge vstd(7)} \qquad \frac{vstd(5):}{e(5,7)\wedge vstd(7)}$$

$$\frac{vstd(7):\neg e(7,1),\neg e(7,2),\neg e(7,3),\neg e(7,4),\neg e(7,5),\neg e(7,6)}{e(7,0)\wedge vstd(0)}$$

$$\frac{vstd(7):\neg e(7,0),\neg e(7,2),\neg e(7,3),\neg e(7,4),\neg e(7,5),\neg e(7,6)}{e(7,1)\wedge vstd(1)}$$

$$\frac{vstd(7):\neg e(7,0),\neg e(7,1),\neg e(7,3),\neg e(7,4),\neg e(7,5),\neg e(7,6)}{e(7,2)\wedge vstd(2)}$$

$$\frac{vstd(7):\neg e(7,0),\neg e(7,1),\neg e(7,2),\neg e(7,4),\neg e(7,5),\neg e(7,6)}{e(7,3)\wedge vstd(3)}$$

$$\frac{vstd(7):\neg e(7,0),\neg e(7,1),\neg e(7,2),\neg e(7,3),\neg e(7,5),\neg e(7,6)}{e(7,4)\wedge vstd(4)}$$

$$\frac{vstd(7):\neg e(7,0),\neg e(7,1),\neg e(7,2),\neg e(7,3),\neg e(7,4),\neg e(7,6)}{e(7,5)\wedge vstd(5)}$$

$$\frac{vstd(7):\neg e(7,0),\neg e(7,1),\neg e(7,2),\neg e(7,3),\neg e(7,4),\neg e(7,5)}{e(7,6)\wedge vstd(6)}$$

$$\frac{:\neg vstd(0),\neg F(0)}{F(0)} \qquad \frac{:\neg vstd(1),\neg F(1)}{F(1)} \qquad \frac{:\neg vstd(2),\neg F(2)}{F(2)}$$

$$\frac{:\neg vstd(3),\neg F(3)}{F(3)} \qquad \frac{:\neg vstd(4),\neg F(4)}{F(4)} \qquad \frac{:\neg vstd(5),\neg F(5)}{F(5)}$$

$$\frac{:\neg vstd(6),\neg F(6)}{F(6)} \qquad \frac{:\neg vstd(7),\neg F(7)}{F(7)} \qquad \frac{:\neg e(7,0),\neg F}{F}.$$

4 Conclusions

In this paper we presented a system, called TheoryBase, to generate test default theories and logic programs. This proposal was motivated by our work on the implementation of Default Reasoning System, which requires extensive experimentation. TheoryBase is based on a graph generating system, the Stanford GraphBase. A wealth of graph examples created in GraphBase induces a large number of examples that can be generated from TheoryBase. Consequently, TheoryBase will greatly facilitate experimental studies of nonmonotonic systems.

However, the number of procedures in TheoryBase is, so far, restricted. In the future, encodings of additional graph problems need to be added to TheoryBase. Moreover, systems for nonmonotonic reasoning are suitable for solving not only problems on graphs and we hope to add to TheoryBase methods to generate theories based on problems in other application domains.

Finally, let us observe that the methodology presented in our paper can be used to produce test cases for other domains. For example, by encoding graph problems as propositional theories, one can obtain a test generating tool to facilitate experimental work on algorithms for satisfiability testing.

References

[1] C. Baral and V.S. Subrahmanian. Dualities between alternative semantics for logic programming and nonmonotonic reasoning. In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *Logic programming and non-monotonic reasoning*, pages 69–86. MIT Press, 1991.

[2] C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian. Implementing stable semantics by linear programming. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning*, pages 23–42. MIT Press, 1993.

[3] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78:85–112, 1991.

[4] B. Bollobás. *Random Graphs.* Academic Press, 1985.

[5] Marco Cadoli, Francesco M. Donini, and Marco Schaerf. Is intractability of non-monotonic reasoning a real drawback? In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 946–951, Seattle, USA, 1994.

[6] J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of AAAI-94*, Menlo Park, CA., 1994. American Association for Artificial Intelligence, Morgan Kaufmann.

[7] M. Dixon and J. de Kleer. Massively parallel assumption-based truth maintenance. In M. Reinfrank, J. de Kleer, M.L. Ginsberg, and E. Sandewall, editors, *Non-monotonic reasoning*, pages 131–142. Berlin: Springer-Verlag, 1989. Lecture Notes in Artificial Intelligence, 346.

[8] M.R. Garey and D.S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[9] M. Gelfond. On stratified autoepistemic theories. In *Proceedings of AAAI-87*, pages 207–211, Los Altos, CA., 1987. American Association for Artificial Intelligence, Morgan Kaufmann.

[10] M.L. Ginsberg. A circumscriptive theorem prover. In M. Reinfrank, J. de Kleer, M.L. Ginsberg, and E. Sandewall, editors, *Non-monotonic reasoning*, pages 100–114. Berlin: Springer-Verlag, 1989. Lecture Notes in Artificial Intelligence, 346.

[11] M.L. Ginsberg and D.A. McAllester. GSAT and dynamic bactracking. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principles of knowledge representation and reasoning, KR '94*, pages 226–237, Cambridge, MA, 1994. Morgan Kaufmann.

[12] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.

[13] U. Junker and K. Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *Proceedings of AAAI-90*, Los Altos, CA., 1990. American Association for Artificial Intelligence, Morgan Kaufmann.

[14] H.A. Kautz and B. Selman. Hard problems for simple default logics. In *Proceedings of the 1st international conference on principles of knowledge representation and reasoning, KR '89*, pages 189–197, San Mateo, CA., 1989. Morgan Kaufmann.

[15] D. E. Knuth. *The Stanford GraphBase: a platform for combinatorial computing.* Addison-Wesley, 1993.

[16] W. Marek, A. Nerode, and J.B. Remmel. Nonmonotonic rule systems II. *Annals of Mathematics and Artificial Intelligence*, 5:229–263, 1992.

[17] W. Marek and M. Truszczyński. Stable semantics for logic programs and default theories. In E. Lusk and R. Overbeek, editors, *Proceedings of the North American conference on logic programming*, pages 243–256, Cambridge, MA., 1989. MIT Press.

[18] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.

[19] W. Marek and M. Truszczyński. *Nonmonotonic logics; context-dependent reasoning.* Berlin: Springer-Verlag, 1993.

[20] J. McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[21] J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[22] D. McDermott. Nonmonotonic logic II: nonmonotonic modal theories. *Journal of the ACM*, 29:33–57, 1982.

[23] D. McDermott and J. Doyle. Nonmonotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[24] R.C. Moore. Possible-world semantics for autoepistemic logic. In R. Reiter, editor, *Proceedings of the workshop on non-monotonic reasoning*, pages 344–354, 1984. (Reprinted in: M.Ginsberg, editor, *Readings on nonmonotonic reasoning.* pages 137–142, 1990, Morgan Kaufmann.).

[25] R.C. Moore. Semantical considerations on non-monotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[26] I. Niemelä. Constructive tightly grounded autoepistemic reasoning. In *Proceedings of IJCAI-91*, pages 399–404, San Mateo, CA., 1991. Morgan Kaufmann.

[27] I. Niemelä. On the decidability and complexity of autoepistemic reasoning. *Fundamenta Informaticae*, 17:117–155, 1992.

[28] C. Papadimitriou and M. Yannakakis. Tie-breaking semantics and structural totality. In *Proceedings of 11th Symposium on Principles of Database Systems*, pages 16 – 22, 1992.

[29] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[30] T. Przymusinski. Autoepistemic logic of closed beliefs and logic programming. In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *Logic programming and non-monotonic reasoning*, pages 3–20. MIT Press, 1991.

[31] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 55–76. Plenum Press, 1978.

[32] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[33] L. Sterling and E. Shapiro. *The Art of Prolog.* Cambridge, MA: MIT Press, 1986.

[34] J. Stillman. The complexity of propositional default logics. In *Proceedings of AAAI-92*, pages 794–799, Menlo Park, CA., 1992. American Association for Artificial Intelligence, Morgan Kaufmann.

[35] D.S. Warren W. Chen, T. Swift. Efficient computation of queries under the well-founded semantics. Technical Report 93-CSE-33, Southern Methodist Univeristy, 1993.