Experimenting with Power Default Reasoning

Eric Klavins and William C. Rounds
Artificial Intelligence Laboratory
University of Michigan
Ann Arbor, Michigan 48109
klavins@umich.edu, rounds@umich.edu

Guo-Qiang Zhang
Dept. of Computer Science
University of Georgia
Athens, Georgia 30602
gqz@cs.uga.edu

Abstract

In this paper we explore the computational aspects of Propositional Power Default Reasoning (PDR), a form of non-monotonic reasoning in which the underlying logic is Kleene's 3-valued propositional logic. PDR leads to a concise meaning of the problem of skeptical entailment which has better complexity characteristics than the usual formalisms (co-NP(3)-Complete instead of Π_2^P-Complete). We take advantage of this in an implementation called `powdef` to encode and solve hard graph problems and explore randomly generated instances of skeptical entailment.

Introduction [1]

Many forms of propositional default reasoning have unexpectedly intractable complexity problems. For example, deciding skeptical entailment in normal propositional default logic (Reiter 1980) is Π_2^P-complete, a result due to Gottlob (Gottlob 1992). Even special cases stay this way – they tend to be NP-hard when we might expect them to be polynomial (Kautz and Selman 1989). This somewhat paradoxical situation has tended to inhibit full-scale use of such reasoning systems.

Recently, however, experimental tests of default reasoning systems have been carried out even in the face of these discouraging complexity results. Perhaps most notably, work with the system DeReS at the University of Kentucky (Cholewiński, Marek and Truszczyński 1994) has shown that it is possible to encode heuristics for graph search problems in a default formalism, and to employ non-monotonic theorem provers to find solutions to these problems (from the Stanford Graph Base (Knuth 1993)) in a relatively efficient manner. (See also (Niemelä and Simons 1997)).

Even more recently, Rounds and Zhang (Zhang and Rounds 1997) introduced a version of Reiter's default logic which they called Power Default Reasoning (PDR). The fundamental difference between PDR and standard default logic is that instead of working with a syntactic representation of knowledge, it uses a *semantic* one based on the tools of *domain theory*, introduced by Dana Scott (Scott 1982) in 1970 as a theory of computation in the semantics of programming languages and developed extensively since then. Rounds and Zhang use default rules to build up partial models instead of using them to "prove" theorems. They can build up *disjunctive* models using a standard construction known as the *Smyth powerdomain*. The concept of *extension*, central to Reiter's logic, is retained in PDR, but it is represented directly as a set of partial models instead of as a theory.

Working this way with models instead of theories improves dramatically on the complexity of nonmonotonic reasoning. For example, while deciding skeptical entailment in standard propositional default logic is Π_2^P-complete, the corresponding problem in PDR is co-NP(3)-complete[2]. This means essentially that a non-deterministic algorithm can decide entailment using at most three calls to an oracle for the satisfiability problem. This result, however, is only theoretical. It provides little information about how such an algorithm could be efficiently implemented, and it leaves open the question as to the distribution of positive and negative instances of entailment – it is perfectly conceivable that almost no instances of the problem would be positive.

In this paper we explore the computational aspects of PDR using a new implementation called `powdef`. We consider propositional logic, and take partial models to be partial truth assignments to the variables of propositional formulae. We use Kleene's strong 3-valued tables to evaluate propositional formulas. Since the co-NP(3) complexity result mentioned above suggests the use of three "oracle" calls to a SAT checker, we give reductions for transforming aspects of power default systems into instances of (2-valued) SAT. Our system has a slot for a SAT checker (we tested several) to find truth assignments. Sets of such assignments are the non-monotonic extensions of a default system; we can use these to check non-monotonic entailment.

We use `powdef` to solve hard graph problems from the Stanford Graph Base. This enables a compari-

[2] A Co-NP(3) problem of the Boolean Hierarchy can be expressed as $(L_1 \cap L_2) \cup L_3$, where L_1 and L_3 are in co-NP, and L_2 is in NP (Cai et al. 1988).

son with DeReS, although we are not really comparing implementations of the same theoretical model, but rather, trying to see if overall run-times are nearly the same on similar instances of graph search problems. We also use different encodings of these search problems. PDR allows us to state instances of a search problem (e.g., Hamiltonian cycle) using disjunctive defaults. This is a natural way to express such problems, but is not straightforward to do in a theory-based system like DeReS. Our running times depend on the kind of SAT checker used, and the class of problem solved. In several cases they are much faster than DeReS, and in a few cases very much slower.

We feel that a simple comparison of our system with DeReS, however, is only part of the experimental story. Since entailment in PDR is co-NP(3)-complete, we decided to check entailment in randomly generated instances of the problem. (This is, in fact, the first general attempt to provide a map of instances of a complete problem in this complexity class.) Our preliminary results show that the entailment problem is remarkably similar to the satisfiability problem, in that hard instances of the problem congregate in special regions, and in that "phase-transition" phenomena can be observed.

In Section 2, we give an overview of PDR. We keep most of this exposition to the finite case (propositional formulae over a finite set of variables), and note that the results hold for the infinite case as well (e.g. first order logic). In Section 3, we describe the translation to SAT. Finally, in Section 4, we present our experimental results.

Power Defaults

In this section we give an overview of PDR. This involves some definitions from domain theory and an introduction to Kleene's 3-valued logic. Domains –in particular, the Scott domains – arise in denotational semantics for programming languages (Scott 1982). In keeping with the idea that a Scott domain provides a semantics for a program, we think of default logic as a kind of logic programming and we will find a natural domain for default rules and extensions. In the rest of this paper we will encounter two Scott domains: the domain of partial truth assignments, which we describe next, and the Smyth powerdomain of a finite domain.

The Domain of Partial Truth Assignments

In our system, the meaning of a propositional formula ϕ will be the set of all partial truth assignments that satisfy it. A partial truth assignment of a formula over a set of variables V is a partial function from V into $\{0,1\}$. For convenience we will use the following definition:

Definition 1 *Given a set of propositional variables V, a* ***truth assignment*** *over V is a function $e : V \rightarrow \{0,1,\perp\}$.*

Here, 0 means false, 1 means true and $\perp$ means undefined. It is customary to denote truth assignments as products or implicants. For example, suppose e is the truth assignment over $\{a,b,c\}$ that maps a to 1, b to $\perp$ and c to 0. Then we write $e = ac'$.

Definition 2 *A* ***satisfier*** *e of a propositional formula ϕ is a truth assignment such that ϕ evaluates to 1 under e. We write $e \models \phi$.*

To evaluate a formula under a partial truth assignment we use Kleene's strong 3-valued logic, specified by the following truth tables:

$\wedge$	**1**	**0**	$\perp$
1	**1**	**0**	$\perp$
0	**0**	**0**	**0**
$\perp$	$\perp$	**0**	$\perp$

$\vee$	**1**	**0**	$\perp$
1	**1**	**1**	**1**
0	**1**	**0**	$\perp$
$\perp$	**1**	$\perp$	$\perp$

p	$\neg p$
1	**0**
0	**1**
$\perp$	$\perp$

Let D be the set of truth assignments over a set of variables V. This is isomorphic to the set of partial functions from V to $\{0,1\}$. We can define a partial order $(D, \sqsubseteq)$ as follows. First, $\{0,1,\perp\}$, the *flat domain*, is ordered $\perp \sqsubseteq 0, \perp \sqsubseteq 1$ and reflexively. Then, if d and e are truth assignments in D, $d \sqsubseteq e$ if $d(v) \sqsubseteq e(v)\ \forall v \in V$. (e.g. $ac' \sqsubseteq ab'c'd$ while ac and ac' are incomparable). Thus, if $d \sqsubseteq e$, then e defines at least all the variables d does and in the same way. We can say that e contains at least as much information as d does and is furthermore in agreement with the information that d gives. In fact, the partial order $(D, \sqsubseteq)$ of partial truth assignments is a Scott domain.

Next, we mention some properties of $(D, \sqsubseteq)$. First, if S is a subset of D, then the *upward closure* $\uparrow S$ of S is the set $\{y \in P | \exists x \in S \text{ such that } x \sqsubseteq y\}$. If $S = \uparrow S$ then S is *upward closed*. $\uparrow\{e\}$ is written $\uparrow e$. An important upward closed subset of the Scott domain of partial truth assignments $(D, \sqsubseteq)$ is the set of all satisfiers of a formula ϕ, written $[\![\phi]\!]$ (this follows from the fact that if $d \sqsubseteq e$ and $d \models \phi$ then $e \models \phi$).

The Smyth Powerdomain

The Smyth powerdomain[3] $P^\sharp(D)$ of a finite Scott domain D is defined to be the set of all nonempty upward closed subsets of D, ordered by reverse subset inclusion. In $P^\sharp(D)$, the ordering $\supseteq$, is commonly denoted by $\sqsubseteq^\sharp$. A disjunction of literals gives rise to an upward-closed set of satisfiers, so the Smyth powerdomain is a natural way to model disjunctive default conclusions. It turns out that the Smyth powerdomain of any Scott domain is again a Scott domain.

Power Defaults

A propositional default system[4] is a triple, $S = (\phi, \Sigma, \psi)$, where ϕ and ψ are propositional formulas called the *initial information* and the *query* respectively, and Σ is a set of *syntactic constraints*. The

[3]For the general case of the Smyth powerdomain and Power Defaults see (Zhang and Rounds 1997).

constraints in Σ have the form $\tau \rightsquigarrow \theta$ where τ and θ are propositional formulas with the approximate meaning: "if τ is known and it is consistent to believe θ, then believe θ". We note for purposes of comparison that in Reiter's notation (Reiter 1980), this rule might be written $\dfrac{\tau : \theta}{\theta}$. We may sometimes refer to a system $S = (\phi, \Sigma)$ when the query is not important.

Extensions

In default logic an extension is a minimal theory that contains the initial information about the world and which is closed under applications of default rules. The insight of Rounds and Zhang is that the fixed-point definition of Reiter carries over almost word for word to elements of a Scott domain. Since the set of partial models of a formula in Kleene logic lives in the Smyth powerdomain, we may use Reiter's definition on the space of partial models, of a set of formulas, rather than on the space of theories.

It is not necessary here to go into the details of how Rounds and Zhang define extensions in the Smyth powerdomain. They show how to "compile" a set Σ of syntactic default rules into a set Δ of semantic (normal) defaults, so that Reiter's definition of extension may be applied. It is, however, important to define non-monotonic entailment, or *skeptical entailment* for a default system:

Definition 3 *Given a system $S = (\phi, \Sigma, \psi)$ we say that ψ is a* ***non-monotonic consequence*** *of ϕ with respect to the set of power defaults Δ constructed from Σ if $[\![\psi]\!] \sqsubseteq^\sharp E$ for every extension E of $[\![\phi]\!]$. We write $\phi \mid\!\sim \psi$.*

Notice how sets of partial models, ordered by reverse inclusion, here replace theories, ordered by inclusion.

We now turn to the concept of *safe* and *unsafe* elements and extensions.

Definition 4 *We say that a truth assignment e over variables in $S = (\phi, \Sigma)$ is* safe *if and only if $e \models \phi$ and for every $\tau \rightsquigarrow \theta \in \Sigma$, if $e \models \tau$ then $e \models \theta$. If a system has a safe extension, it is said to be safe.*

The key to the Rounds-Zhang co-NP(3) complexity result, which also enables our implementation, is the following theorem.

Theorem 1 ***(Dichotomy Theorem)*** *For a system $S = (\phi, \Sigma, \psi)$, either the set of partial models of ϕ has a unique safe extension, or else all extensions of this set are unsafe, and are sets containing a single, total truth assignment g (one that doesn't assign $\perp$ to any variable) such that $g \models \phi$.*

[4] Although we are only concerned with propositional default systems in this paper, first order logic or any other logic may be used as long as there is a way to express the semantics of the logic as a Scott domain.

Remarkably, we then have that for a system $S = (\phi, \Sigma, \psi)$ that $\phi \mid\!\sim \psi$ if there is a safe element and $e \models \psi$ for all safe elements (we call this *safe entailment*), or if the system is not safe and $\phi \rightarrow \psi$ is a (classical) tautology. This implies that only 3 NP "oracle" calls are needed to decide full skeptical entailment for a propositional system. See (Zhang and Rounds 1997) for the details of this complexity result.

Computing Skeptical Entailment

Translation to 2-Valued SAT

We would like to be able to compute the safe extensions of a system since doing so would solve the safe entailment problem and safe extensions may be interesting in their own right. One might be tempted to construct a new formula for the system by considering the defaults as implications, conjoining the initial formula and finding 2-valued satisfiers for it. That is, consider $\phi \wedge (\tau_1 \rightarrow \theta_1) \wedge \ldots \wedge (\tau_n \rightarrow \theta_n)$. But in 3-valued logic, $e \not\models \tau$ is not the same as $e \models \neg\tau$ so this simple construction fails. Nevertheless, given $S = (\phi, \Sigma)$ we can construct a 2-valued formula Φ such that there is a one-to-one correspondence between the satisfiers of Φ and the safe elements of S. To do this we first assume that our formulas are in Negation Normal Form (NNF), where negations appear only on atoms. The NNF of a formula can be found in polynomial time by Demorgan's laws. It is necessary for the proof of Proposition 1, which is by induction on the structure of the formula. We then introduce, for each of the n variables i occurring in S, two new variables P_i and N_i with the intended meaning that P_i is 1 only when variable i is 1 and N_i is 1 when variable i is 0 and P_i and N_i are both 0 only when variable i is $\perp$. Next we construct a sub-formula Ψ of Φ as follows:

$$\Psi = \bigwedge_{i=1}^{n} (\neg P_i \vee \neg N_i).$$

This formula limits the range of the 2-valued satisfiers we will consider by mimicking the 3-valued world of S. Clearly, any satisfier of Ψ will not assign 1 to both P_i and N_i.

Next, for an arbitrary 3-valued NNF formula α, let $R(\alpha)$ be the formula that results from replacing positive occurrences of variable i in α with P_i and negative occurrences of variable i in α with N_i for all i.

Proposition 1 *Let d be a 3-valued truth assignment for a NNF formula α and let $f(d)$ be the satisfying 2-valued truth assignment of Ψ constructed for variables in α that assigns P_i 1 if variable i is 1 in d, N_i 1 if variable i is 0 in d, and N_i and P_i both 0 if variable i is $\perp$ in d. Then $d \models \alpha$ if and only if $f(d) \models \Psi \wedge R(\alpha)$.*

Note that f is one-to-one. This implies that whenever we have a 2-valued satisfier for $\Psi \wedge R(\alpha)$ we may always reconstruct a 3-valued satisfier for α. Also note that R preserves connectives. Finally, our desired two-valued formula Φ is

$$\Psi \wedge R(\phi) \wedge \bigwedge_{\tau \leadsto \theta \in \Sigma} R(\tau) \rightarrow R(\theta).$$

Proposition 2 *Let e be a truth assignment over the variables in $S = (\phi, \Sigma)$, Φ be the 2-valued formula constructed from S as above, and $f(e)$ be a satisfier of Ψ constructed as in Proposition 1. Then e is a safe element of S if and only if $f(e) \models \Phi$.*

Notice that by these two propositions, "$e \models \psi$ for all safe elements" is equivalent to "$\Phi \rightarrow R(\psi)$ is a tautology".

Experiments

The Implementation

Our implementation of safety checking and safe entailment is a program written in C called `powdef`. Given a system S as input, `powdef` constructs a CNF formula equivalent to Φ as above and then uses a complete SAT checker to check whether or not Φ is satisfiable (SAT checkers usually prefer their input to be in CNF). If Φ is satisfiable (so that S is safe), `powdef` checks that $\Phi \rightarrow R(\psi)$ is a tautology. Otherwise, `powdef` checks that $\phi \rightarrow \psi$ is a tautology. If the tautology check succeeds `powdef` returns "yes", otherwise it returns "no". Note that with a SAT checker we can check that a formula $\alpha \rightarrow \beta$ is a tautology by checking that $\alpha \wedge \neg\beta$ is *not* satisfiable.

If the action specified to `powdef` is not to check entailment but to find minimal safe elements, a list of safe elements found so far is kept. When a new one is found, old safe elements that are greater than the new one, in the Scott domain ordering, are thrown out. If a new satisfier is found to be greater than one already found, it is not added to the list. In practice, this method has proved the most efficient of those we tried, which included conjoining expressions to Φ to limit satisfiers to those that corresponded to minimal safe elements. The addition of such formulas usually resulted in prohibitively many CNF clauses.

The SAT checker we use in most of our experiments uses the simple Davis-Putnam procedure for CNF-SAT (Davis and Putnam 1960), modified to return all satisfiers. We compared several SAT checkers including Selman's randomized SAT checker WalkSAT (Selman, Kautz and Cohen 1993).

A Comparison with an Existing Implementation

In this section we try to compare `powdef` to DeReS (Cholewiński, Marek and Truszczyński 1994). DeReS uses Reiter's default theories, so this comparison is tenuous at best.

Expressing Hard Graph Problems The DeReS implementation uses a subsystem TheoryBase which converts instances of graph problems, such as COLORABILITY, HAMILTON CIRCUIT or KERNELS into instances of default theories, the extensions of which are solutions to the problems. The graphs are generated using the Stanford Graph Base (Knuth 1993). Since we do not use theories, we do not use TheoryBase. We do, however, use the Stanford Base to generate instances. It is easy to represent instances of such problems as power default systems.

Example 1 *Finding Kernels* Let $G = (V, E)$ be a directed graph with n vertices. A *kernel* in G is a subset K of V such that no two vertices in K are joined by an edge in E and such that for every $v \in V \setminus K$ there is a $u \in K$ such that $(u, v) \in E$. A default system $S(G)$ is constructed such that G has a kernel if and only if $S(G)$ has a safe element. The minimal safe elements of the system will correspond exactly to the kernels of the graph. Let $K(i)$ represent the fact that node i is in the kernel for $1 \leq i \leq n$. We use ϕ to seed the kernel by saying each node is either in or not in the kernel:

$$\phi = \bigwedge_{1 \leq i \leq n} K(i) \vee \neg K(i)$$

To specify the conditions under which a node is in a kernel we use default rules. For each $1 \leq i \leq n$ we have the two default rules

$$K(i) \leadsto \bigwedge_{(i,j) \in E} \neg K(j) \;\; \textit{and} \;\; \neg K(i) \leadsto \bigvee_{(i,j) \in E} K(j).$$

The first states that, normally, if a node is in a kernel then none of its neighbors are. The second states that, normally, if a node is not in a kernel then at least one of its neighbors is.

Note that a kernel is only completely specified if each $K(i)$ is defined as either 1 or 0. Thus, all safe elements are total truth assignments over the n variables and we need only find the 2-valued satisfiers of the system considered as the 2-valued formula

$$\phi \wedge \bigwedge_{\tau \leadsto \theta \in \Sigma} (\tau \rightarrow \theta).$$

This means that kernels are not an adequate test of our three-valued model, so we turn to another example.

Example 2 *k-Coloring* Let $G = (V, E)$ be an undirected graph with n vertices. A *coloring* of G with k colors is an assignment of one of k colors to each node of G such that no two adjacent nodes have the same color. We construct default system $S(G)$ such that G is k-colorable iff $S(G)$ has at least one safe element. The minimal safe elements, if there are any, are truth assignments that represent possible k colorings of G. The construction of $S(G)$ is as follows. We choose propositional variables $C(i, c)$ for $1 \leq i \leq n$ and $1 \leq c \leq k$ to represent the fact that node i is colored with color c.

m	vars	clauses	sols	1 solution				all solutions	
				DP	POSIT	WalkSAT	DeReS	DP	DeReS
2	16	81	2	0.01	0.001	0.001	0.01	0.01	0.02
4	32	161	6	0.02	0.001	0.001	0.07	0.03	0.32
6	48	241	5	0.02	0.03	0.002	0.57	0.08	1.90
8	64	321	134	0.03	0.002	0.016	1.45	1.16	18.97
10	80	401	267	0.04	0.003	0.017	9.14	3.69	141.74

Figure 1: finding kernels in 8xm knight's wrapped chess boards

n	DP	POSIT	WalkSAT	DeReS
200	10.73	0.541	0.147	0.18
400	40.78	2.235	0.291	0.34
600	-	5.181	0.441	0.50
800	-	9.156	0.664	0.68
1000	-	14.666	0.182	0.84

Figure 2: coloring on a 2xn ladder, one solution — a "-" indicates a CPU time greater than two minutes

The initial formula ϕ states that each node must have a color,

$$\phi = \bigwedge_{i=1}^{n} \bigvee_{c=1}^{k} C(i,c)$$

and each default rule states roughly that, normally, if a node is colored with a color c then none of its neighbors are colored with color c. Note that if a node has more than one color then it can be assigned any of those possible colors. Thus, the defaults are

$$C(i,c) \rightsquigarrow \bigwedge_{\{i,j\}\in E} \neg C(j,c)$$

for each $1 \leq i \leq n$ and $1 \leq c \leq k$.

Here, since the defaults allow for a node to be colored with any color that is consistent with a coloring, the minimal safe elements are not complete truth assignments and we must use the reduction described in the previous section.

Experimental Results For this experiment, we used the Stanford Graph Base to give us instances of KERNELS and 3-COLORING. In general, the problems were generated and encoded as CNF-SAT and then solved in turn with the various SAT checkers discussed above to compare performance. None of the SAT checkers is necessarily optimized for finding all solutions, as SAT checkers are usually optimized for decision problems. Our tables report CPU times in seconds on a Sun Ultra. Times for DeReS on the same problems are quoted from (Cholewiński, Marek and Truszczyński 1994); keep in mind that we are not comparing implementations of the same model here, although we are solving the same graph problems.

Figure 1 shows results for the kernel problem applied to the class of graphs generated by 8xm knight's wrapped chess boards for $m = 2,4,6,8,10$. Since the solutions are total, this example was simply coded as the conjunction of ϕ and all the default rules considered as implications, and solved for minimal three-valued satisfiers. The time for one solution (safe element) using our simple implementation of the Davis-Putnam method (Davis and Putnam 1960), Jon Freeman's POSIT (Freeman 1995), and Bart Selman's random hill-climbing WalkSAT (Selman, Kautz and Cohen 1993) are shown. Only the Davis-Putnam procedure was used in finding all safe elements. In this example we see the most dramatic improvement since the translation to 2-valued SAT is not needed.

Figure 2 shows results for the 3-coloring problem applied to 2xn ladders (graphs that can be laid out to look like ladders). Since there are a large number of colorings for a ladder we only report the time to find one safe element. Note that DP and POSIT perform poorly on this problem because of the large number of variables. Because it is randomized, WalkSAT consistently performs better on problems of this type.

The results of this section are promising, but we feel that graph problems do not capture the essence of nonmonotonicity. In the colorability problem, for example, we say that normally, a node can't be colored the same way as any of its neighbors. In fact, though, a node can *never* be colored the same way as one of its neighbors. For general graph problems, solutions correspond to minimal safe elements, and there are no exceptions. That is, if e is safe and $e \sqsubseteq f$, then f is also safe. This is not generally the case: a system with exceptions will have *rogue* elements – unsafe elements f with a safe element e with $e \sqsubseteq f$. For example, consider the simple system:

$$S = (tweety, \{bird \rightsquigarrow fly, tweety \wedge bird \rightsquigarrow \neg fly\}).$$

The minimal safe element of S is the assignment e that only defines $tweety$ to 1. The assignment f that assigns

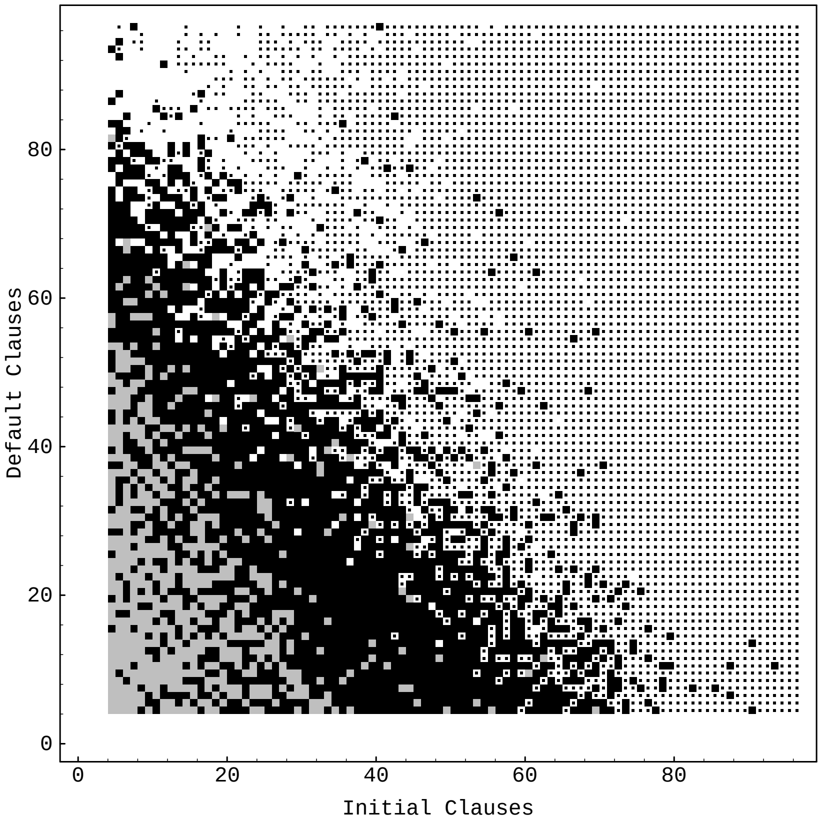

Figure 3: $v = 20, q = 5, i$ and d varied. For each coordinate, (i, j) a random instance with i initial clauses and j defaults was checked. Gray squares are safely not entailed, black are safely entailed, white are unsafe instances and sqaures with a dot are unsafe but entailed.

tweety to 1 and *bird* to 1 is not safe but is such that $e \sqsubseteq f$, so f is a rogue.

Given these observations, we experimented with random instances of default systems, to see the general layout of the skeptical entailment problem.

Randomly Generated Instances

Our experiment in this section is inspired by the SAT community's exploration of randomly generated 3-SAT problems. The following parameters determine the nature of a randomly generated default system:

- v = Number of variables
- i = Number of 3 literal clauses in ϕ
- d = Number of defaults in Σ of the form $a \rightsquigarrow b \vee c$
- q = Number of literals in ψ

where ϕ is a CNF formula with i clauses and ψ is a disjunction of q (positive or negative) literals.

The experiment consisted of fixing v at 20 variables, the number of literals in the query at $q = v/4 = 5$ and varying i and d from 4 to 120 to get an idea of the dependence of safety and safe entailment on i and d. Thus, 116×116 random instances were generated and checked for skeptical entailment. Figure 3 shows the results of this experiment. Note that for i and d small, the system is under-specified, and thus, there are many safe elements. In these cases, the probability that ψ will be satisfied by all of them is low. As i and d increase, there are fewer safe elements and ψ is more likely to be entailed. As $i + d$ exceeds about 70 to 75, the systems become over-specified and are more often unsafe. Remarkably, systems seem to become unsafe approximately when $i + d = 4.3v = 68.8$. The number 4.3 is the well-known phase transition point for the 2-valued SAT problem (Mitchell, Selman and Levesque 1992). This intriguing similarity with random 3-SAT problems deserves more exploration.

We omit details of CPU times for this experiment and simply note that in the large majority of instances the CPU time required to decide skeptical entailment was small (under a second). The times were the greatest when i and d were small and the system was under-specified. As the number of safe elements decreased, so did the CPU time needed. It seems likely that in an application of power defaults, it would not be difficult to keep the system from being under-specified. That CPU times for i and d small tend to be large is probably due to the fact that with under-specified systems there are (many) more safe elements to check. Experimentation with a more direct algorithm (not given here) also suggests that CPU times for i and d small may not be subject to much improvement.

Whether or not a query is entailed in a system depends on the "liberalness" of the query. The number of entailed instances in a large sample of random instances grows with the number of literals in the query (a disjunction of literals). The query size also affects the liklihood that $\phi \rightarrow \psi$ will be a tautology. Notice in Figure 3 that unsafe, entailed instance (small dots) become less abundant after a certain point. Where this happens corresponds to the query size. A less liberal query would move this point to the right. Also notice that whether or not these instances are entailed is independent of the defaults.

The results of this section tell us that safety is not a completely elusive property, which is exciting. If it is likely that a system is safe, then checking for safety is not a wasted exercise because we can find safe elements and then check entailment. Our results do show that a fair number of systems actually have only one extension (the safe one, by the Dichotomy Theorem).

Conclusion

We conclude by noting some areas of the work that seem worthy of further exploration. We described an indirect method of computing extensions based on a translation to SAT. We are currently testing a more direct algorithm – a three-valued generalization of the Davis-Putnam procedure. We hope that such an algorithm may be extended to the case of first-order logic.

PDR can express graph problems very naturally. However, we noted that these problems, while good for testing non-monotonic systems, are not essentially non-monotonic themselves. It is difficult to say what a real-life large-scale non-monotonic problem is; likely candidates might be found in the areas of planning, diagnosis and synthesis. It may be that a formal distinction based on the presence or absence of rogue elements can be made.

Ginsberg (Ginsberg 1991) suggests that default rules can be used as computational aids by providing soft rules (islands) that limit the search space of a problem — even though the problem may be solvable without the rules. This seems to be a good general strategy, according with Dichotomy Theorem. That is, if the default rules tell you something consistent and usable, then use it; otherwise, decide entailment without the default rules. This direction may also lead to algorithms for *learning* default rules from examples.

Finally, some theoretical questions; (1) Can we treat non-normal defaults? (2) Can we consider defaults in CPO's besides Scott domains? (3) How expressive is the non-disjunctive case of PDR? For this case entailment is a low-order polynomial-time problem (Zhang and Rounds 1997).

References

J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, "The Boolean Hierarchy I: Structural Properties", SIAM Journal of Computing, 17, 1232-1252, 1988.

P. Cholewiński, V. W. Marek and M. Truszczyński, "Default Reasoning System DeReS", *in Proceedings of KR-96*, 1996.

M. Davis and H. Putnam, "A computing procedure for quantification", *J. Assoc. Computing Mach.*, 7, 201-215, 1960.

J.W. Freeman, "Improvements to Propositional Satisfiability Search Algorithms", Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania, May 1995.

M.L. Ginsberg, "The Computational Value of Nonmonotonic Reasoning", *Proceedings of KR-91*, 1991.

G. Gottlob, "Complexity results for non-monotonic logics", *J. Logic and Computation*, 2(3):397-425, 1992.

H. Kautz and B. Selman, "Hard Problems for Simple Default Logics", *in Proceedings of KR-89*, 1989.

D.E Knuth, *The Stanford Graph Base: a platform for combinatorial computing*, Addison-Wesley, 1993.

D. Mitchel, B. Selman, H.J. Levesque, "Hard and easy distributions of SAT problems", *in Proceedings of AAAI-92*, San Jose, CA, 459-465, 1992.

I. Niemelä and Patrik Simons, "Smodels – and Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs", *in Proceedings of LPNMR-97*, 1997.

B. Selman, H.A. Kautz, B. Cohen, "Local Search Strategies for Satisfiability Testing", Presented at the *Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*, Oct. 1993.

R. Reiter, "A logic for default reasoning", *Artificial Intelligence*, 13:81-132, 1980.

G.-Q. Zhang and W. Rounds, "Complexity of power default reasoning", *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, 328-339, 1997.

D.S. Scott, "Domains for denotational semantics", In *Lecture Notes in Computer Science 140*, 577-613, 1982.