Nonmonotonic Reasoning with Quantified Boolean Constraints

Chris Pollett[1] and Jeffrey B. Remmel[2]

[1] Department of Mathematics, University of California at San Diego, La Jolla, CA 92903, e-mail: cpollett@math.ucsd.edu

[2] Department of Mathematics, University of California at San Diego, La Jolla, CA 92903. Currently with Sagent Corporation, Bellvue, WA, 98004, e-mail: jremmel@sagent.com

Abstract. In this paper, we define and investigate the complexity of several nonmonotonic logics with quantified Boolean formulas as constraints. We give quantified constraint versions of the constraint programming formalism of Marek, Nerode, and Remmel [15] and of the natural extension of their theory to default logic. We also introduce a new formalism which adds constraints to circumscription. We show that standard complexity results for each of these formalisms generalize in the quantified constraint case. Gogic, Kautz, Papadimitriou, and Selman [8] have introduced a new method for measuring the strengths of reasoning formalisms based on succinctness of model representation. We show a natural hierarchy based on this measure exists between our versions of logic programming, circumscription, and default logic. Finally, we discuss some results about the relative succinctness of our reasoning formalisms versus any formalism for which model checking can be done somewhere in the polynomial time hierarchy.

1 Introduction

The motivation for this paper arose naturally from the theory of constraint programs and constraint models as proposed by Marek, Nerode, and Remmel [15]. That is, Marek, Nerode, and Remmel extended the logic programming with negation formalism to encorporate arbitrary constraints. The constraints considered by Marek, Nerode, and Remmel are not restricted to statements on real numbers as in CLP($\mathbf{R}$), see [12]. Instead, they defined a general notion of constraint programs and constraint models and showed that stable models of logic programs as well as the supported models of logic programs are just special cases of constraint models of constraint programs.

In the Marek, Nerode, Remmel theory, the constraint of a clause is not restricted to be of a certain form or even to be expressible in the underlying language of the logic program. A constraint program clause is of the form

$$p \leftarrow a_1, \ldots a_n : \Psi$$

where no particular restrictions were placed on the formula Ψ. The formula Ψ is called a constraint formula. The idea is that the constraint formula of a clause

incorporates the conditions that need to be satisfied by (some) parameters before we start to evaluate the remaining goals in the body of the clause. Thus the constraint controls the applicability of the clause, i.e. , if it fails, then the rule cannot fire. The novel feature of their theory is that the constraints are supposed to be true in the model consisting of the atoms computed in the process. That is, one allows for an *a posteriori* justification of the rules applied in the process of computation. At first glance this is a different phenomenon than occurs in constraint logic programming. There the constraints are applied *a priori*. Indeed, in constraint logic programming, once one finds the multidimensional region containing the solutions of the constraint, the subsequent steps can only refine this description in that the solutions must come from that region. The constraints allowed in a constraint program may be very diverse and complex formulas. They may be formulas in the underlying language of the program or they may be formulas in second order or even infinitary logic. An example of this type of constraint is a parity example where the constraint is a statement about the parity of the (finite) putative model of the program. This type of the constraint is a formula of infinitary logic if the Herbrand base is infinite. The only requirement for a constraint is that there is a method to check if the constraint holds relative to a possible model.

The motivation for allowing more general constraints originally came from the certain applications in control theory of real-time systems. The basic idea is that we sample the plant state at discrete intervals of time, $\Delta, 2\Delta, \ldots$. Based on the plant measurements, a logic program will compute the control to be used by the plant for the next Δ seconds so that the plant state will be guaranteed to meet certain required specifications. One possible way for such logic programs to operate is that the set of rules with which we compute at any give time $n\Delta$ is a function of the observations of the state of the plant at time $n\Delta$. In this fashion, we can view the plant state at time $n\Delta$ as determining which constraints of the rules of the logic program are satisfied and hence which rules can fire at time $n\Delta$. In such a situation, we cannot expect that we will have the constraints which are necessarily in the same language as the underlying language of the program. For example, Ψ could be a formula that asserts that a particular integral has value one. Therefore it seems necessary to step out of the current constraint logic programming paradigm. Another application of the same type is to have a logic program controlling inventory via a deductive database. In this case, we may want to change the rules of the deductive database as a function of outside demand. Once again one could view the satisfaction of the constraints as depending on the database for the inventory and the set of orders that come in at a given time. In this way one can vary the set of applicable rules as function of the current inventory and demand.

The constraint formulas control the applicability of a clause during program evaluation. They allow for an increased flexibility in the forms of data that can be handled by logic programs. The actual evaluation of these constraint formulas might be handled by specialized hardware and so model search for these programs is potentially feasible. Nevertheless, the evaluation of constraint models

which are the natural extension of the stable logic semantics to programs built out of such clauses requires checking these constraints. Hence the checking of constraints becomes a new source of complexity in the the evaluation of constraint models. Thus to study the complexity of this type logic programming with generalized constraints, or indeed any form of nonmonotonic logic with generalized constraints, it is useful to develop constraint theories of various complexity.

Quantified propositional formulas provide a natural tool for developing constraint theories of varying complexity. This is because restricting the quantifier depth of such formulas yields natural complete problems for various levels of the polynomial hierarchy. From the practical point of view quantified Boolean formulas as constraints allow one to check global properties of a database or plant in a way which would be much more cumbersome if one had to rely on propositional constraints alone.

In this paper, we define and investigate the complexity of several nonmonotonic logics with quantified Boolean formulas as constraints. In particular, we give quantified constraint versions of constraint programming of Marek, Nerode, and Remmel [15] and the natural extension of their theory to default logic. We also introduce a new formalism which adds constraints to circumscription. We show that standard complexity results [2, 14] for each of these formalisms generalize in the quantified constraint case. If our constraint formulas are restricted to be QBF_k, quantified Boolean formulas with k alternations of quantifiers, then model existence problem in constraint programming (LP_k) is Σ^p_{k+1}-complete. Similarly, the model existence problem for our version of default logic (DL_k) is Σ^p_{k+2}-complete. For our version circumscription (CC_k) we show that it is Π^p_{k+2}-complete to determine if a given variable is in all models of a circumscribed program. The fact that completeness results for these theories lift so uniformly up through the polynomial hierarchy, i.e. our version of logic programming always remains one level below our versions of circumscription and default logic, illustrates the crucial role that constraints play in complexity results for nonmonotonic theories.

A new method for comparing the relative strength of reasoning formalisms was developed by Gogic, Kautz, Papadimitriou, and Selman [8] (see also Papadimitriou [17] and Gogic [9]). It is based on the succinctness with which a formalism can express a particular collection of models. The idea is that given any reasoning formalisms A and B, we say that A *is at least as representationally succinct as* B, written $B \leq_s A$, if for each knowledge base ϕ_B in the reasoning formalism B, there is a knowledge base ϕ_A in the reasoning formalism A, whose size is polynomial bounded in the size of ϕ_B such that ϕ_B and ϕ_A are defined over the same *free* variables and have the same set of models. Note that there is no requirement that ϕ_A be effectively computed from ϕ_B. It is easy to see succinctness criteria induce a transitive, reflexive relation $\leq_s$ on the class of reasoning formalisms. Gogic , Kautz, Papadimitriou, and Selman showed that there is a rather remarkable hierarchy that results among various reasoning formalisms such as propositional logic, horn logic, circumscription, and default logic given that the polynomial time hierarchy does not collapse at low levels.

We shall show that there is a natural hierarchy that exists among the theories LP_k, DL_k, and CC_k introduced in this paper. We show that $LP_k <_s DL_k$ and $LP_k^* <_s CC_k <_s DL_k$. The strictness of these inclusions is under the assumption that the polynomial hierarchy does not collapse. LP_k^{sup} is LP_k where we consider supported models of programs rather than stable ones. LP_k^* is LP_k restricted to programs whose supported models are pairwise incomparable. With a slightly weaker notion of succinctness $\leq_{ws}$ where one does not insist that ϕ_A and ϕ_B have the same set of variable, one can show:

$$LP_k^* \equiv_{ws} LP_k <_{ws} CC_k \leq_{ws} DL_k \equiv_{ws} LP_{k+1}.$$

We also show that $LP_k \leq_{ws} LP_k^{sup}$ and $LP_k \leq_s LP_k^{sup}$. Thus $LP_k \equiv LP_k^{sup}$. Similar results are proven for default logic and circumscription.

The results of this paper are presented in four sections. In the first three sections, we develop our versions of logic programming, circumscription, and default logic. In the fourth section we discuss our succinctness results. Besides the above set of inclusions, in the fourth section, we discuss some results about the relative succinctness of our reasoning formalisms versus any formalism for which model checking can be done somewhere in the polynomial time hierarchy.

Finally, we wish to thank Victor Marek for carefully reading our first draft to the paper and for many suggested improvements.

2 Logic Programming

We will be considering clauses of the form

$$p \leftarrow a_1, \ldots, a_n : B_1(\boldsymbol{b}_1), \ldots, B_n(\boldsymbol{b}_m) \qquad (*)$$

where $p, a_1, \ldots, a_n$ are propositional variables and B is a quantified Boolean formula[16]. For each i, the $\boldsymbol{b}_i$ represents the free propositional variables in B_i. We call p the *conclusion* of the clause, we call the a_i's the *premises* of the clause, and we call the B_i's *constraint formulas*.

A *logic program with quantified Boolean constraints* is a collection of clauses of the above type. We will often call this just a *constraint logic program* or *program*. The *Herbrand Universe* for such a program consists of all propositional variables appearing free (unquantified) somewhere in the program. Let P be a program. Let M be a subset of the Herbrand Universe of P and let ν_M be the truth assignment such that $\nu_M(x) = 1$ if and only if $x \in M$. We denote by P_M the logic program obtained from P by deleting those clauses whose constraints are not satisfied by ν_M and by deleting the constraints from the remaining clauses. P_M is a Horn program and has a least model N_M. We say that M is a *stable model* of P if it is the case that $M = N_M$.

Every truth assignment ν uniquely extends to a truth assignment $\bar{\nu}$ which evaluates all the quantified Boolean formulas and satisfies the usual Tarskian conditions for satisfaction. A *supported model* for a logic program is a truth assignment ν to its Herbrand Universe such that for each variable p, $\nu(p) = 1$

if and only if there is a clause of type $(*)$ such that $\nu(a_1) = 1, \ldots \nu(a_n) = 1$, and $\bar{\nu}(B_1(\boldsymbol{b_1})) = 1, \ldots \bar{\nu}(B_m(\boldsymbol{b_m})) = 1$. Otherwise, $\nu(p) = 0$. The clause $p \leftarrow$: can be used to force $\nu(p)$ to be 1. We say that a is *in* a model ν if $\nu(a) = 1$. If $\nu(a) = 0$ then a is *out* of ν. Every stable model of P is a supported one, but the reverse need not be true. We will use supported models when we prove our succinctness results. Unless we state otherwise, however, when we say *model* of a logic program we mean *stable model*.

We write Σ_k^q to denote the set of quantified Boolean formulas with at most k-alternations of quantifier type and whose outermost quantifier is an $\exists$. Similarly, we write Π_k^q denote the set of quantified Boolean formulas with at most k-alternations of quantifier type and whose outermost quantifier is an $\forall$. In both cases, unless we say we are dealing with only sentences, we assume our formulas have free variables. Lastly, we write QBF_k to denote Boolean combinations of these two classes. In the $k = 0$ case all of the above classes are the same: The class of propositional formulas. We recall that the problem of determining whether a Σ_k^q-sentence is true is Σ_k^p-complete and the problem of determining whether a Π_k^q-sentence is true is Π_k^p-complete. Given an assignment ν to the free variables of a QBF_k formula B, the problem of determining whether or not $\bar{\nu}(B)$ is true is in Δ_{k+1}^p.

Definition 1. *LP_k is the class of* finite *logic programs whose constraints are all in QBF_k. $LP_\infty = \cup_{k \geq 0} LP_k$.*

Let us pause a moment here to compare the class LP_0 to logic programs that allow for yet simpler constraints. Some results on such programs were proved by Marek, Nerode, and Remmel in [15]. Specifically, for programs with clauses that have propositional constraints they construct an equivalent program in the same language where all the constraints are conjunctions of literals. The reason why this is possible is that constraints distribute with respect to disjunction. That is, a program transformation where replace a clause $p \leftarrow a_1, \ldots a_n : B_1 \vee B_2$ by two clauses, $p \leftarrow a_1, \ldots a_n : B_1$ and $p \leftarrow a_1, \ldots a_n : B_2$, does not change semantics. Consequently, all we need to do is to put constraints in disjunctive normal form (disjunction of conjunctions) and then distribute. We repeat it for all clauses of P if necessary.

This conversion yields a program potentially exponential in the size of the original program. If however one allows for additional variables then we can find a polynomial transformation. We will outline it below. First, by taking the conjunction of all constraints, we can assume that each clause has just one constraint. Indeed, each of the constraints of a clause is satisfied if and only if their conjunction is satisfied. Next, since we do not changes the set of stables of a program if we replace any constraint in the program by a logically equivalent constraint, we can eliminate implications and push the negations downwards to the level of literals. Thus we can assume that the constraints are built out of literals by means of conjunctions and alternatives. Since the constraints split with respect to alternatives, all we need to do is to handle conjunctions. We do

this as follows. Given a clause of the form

$$C = p \leftarrow a_1, \ldots, a_m : \mathbb{\wedge}_{j=0}^{r} B_j$$

we introduce r new constants $b_1, \ldots, b_r$, eliminate C and put into P

$p \leftarrow a_1, \ldots, a_m, b_1, \ldots, b_n :$
$b_1 \leftarrow : B_1$
$\ldots$
$b_n \leftarrow : B_n$

After the execution of these operations, the resulting program P^* (clearly polynomial in size of P) has the following properties. First, every stable (constraint) model of P uniquely extends to a model of P^*. Second, all the constraints of clauses of P^* are either tautologies, or a single literals. We notice that in the context of general logic programs this result is well-known. In the terminology of Section 5 we proved that for the class LP^* of programs with a single constraint which is a tautology or a literal, we have $LP_0 \leq_{ws} LP^*$ (and consequently $LP_0 \equiv_{ws} LP^*$ as well).

Theorem 1.
1. The problem $\exists CMLP_k$ of determining whether an LP_k program has a model is Σ_{k+1}^p-complete.
2. The problem $\exists CMLP_\infty$ of determining whether a finite LP_∞ program has a model is $PSPACE$-complete.

Proof. First, we show $\exists CMLP_k \in \Sigma_{k+1}^p$. Let $P \in LP_k$. We guess a model for P then check which of P's clauses have their constraint formulas satisfied. As we said above, given a truth assignment, checking a QBF_k formula can be done in Δ_{k+1}^p. We delete the clauses whose constraint formulas are not satisfied. The remaining clauses have their constraint formulas satisfied so we can delete the constraint formulas for these formulas. The resulting program is a Horn program and finding a minimal model for a Horn program can be done in linear time [3, 14]. Thus, this whole algorithm can be done in $NP(\Delta_{k+1}^p)$. Therefore, $\exists CMLP_k \in \Sigma_{k+1}^p$.

For completeness, let $B(b_1, \ldots, b_m) \in QBF_k$. Consider the following logic program P^*. It has four groups of clauses:

(1)	$b_i \leftarrow : \neg \bar{b_i}$	$i = 1, \ldots, m$
	$\bar{b_i} \leftarrow : \neg b_i$	
(2)	$\gamma \leftarrow b_i, \bar{b_i} :$	$i = 1, \ldots, m$
(3)	$b_i \leftarrow : \neg\gamma$	$i = 1, \ldots, m$
	$\bar{b_i} \leftarrow : \neg\gamma$	
(4)	$\gamma \leftarrow : B(b_1, \ldots b_m)$	

Because of clauses (2) and (3), P^* has a model M only if $\gamma \in M$. However γ can not be derived by a clause of type (2) since clauses of type (3) only fire if we

do not have γ and clauses of type (1) will only derive one of b_i or $\bar{b_i}$. Hence, γ must be derived by (4). Thus, to derive γ we must be able to solve the problem of whether $\exists \boldsymbol{b} B(\boldsymbol{b})$ is valid. But determining whether a Σ^q_{k+1} formula is valid is a Σ^p_{k+1}-complete problem. Thus $\exists CMLP_k$ is Σ^p_{k+1}-complete.

The LP_∞ case is proven in a similar fashion.

Theorem 2.
1. The problem of deciding whether a given variable a is in a model of an LP_k program is Σ^p_{k+1}-complete.
2. The analogous problem for LP_∞ it is $PSPACE$-complete.

Proof. The problem is in Σ^p_{k+1} since we can just guess a truth assignment ν such that $\nu(a) = 1$ and check that it is a model. To see that our problem is Σ^p_{k+1}-complete notice that the problem of whether γ is in an extension of P in Theorem 1 is equivalent to whether a Σ^q_{k+1}-formula is valid.
The LP_∞ case is similar.

Remark 1. It is not hard to generalize the notion of logic programming with quantified Boolean constraints: One needs only generalize the notion of quantified Boolean formula. For instance, rather than take our atoms of quantified Boolean formulas to be just propositional variables we could let them be propositional variables and expressions of the form $a_1 a_2 \ldots a_n \in A$. That is, checking if the concatenation of some string propositional variables is in an oracle A. Given an assignment ν, we define $\bar{\nu}(a_1 a_2 \ldots a_n \in A) = 1$ if and only if the string $\nu(a_1)\nu(a_2)\ldots\nu(a_n)$ is in the set A. Thus, there is a well defined semantics for such formulas. So we can define the classes $\Sigma^q_k(A)$, $\Pi^q_k(A)$, and $QBF_k(A)$ and use them in our logic programming theories. Hence, we can define $LP_k(A)$ to be finite logic programs with $QBF_k(A)$ constraints. From Goldsmith Joseph [10] and Schöning [18] it is known that $SAT_k(A)$ is $\Sigma^p_{k+1}(A)$-complete. Hence, using exactly the same arguments we can generalize the theorems we have just obtained for LP_k.

Theorem 3.
1. The problem of deciding whether an $LP_k(A)$ program has a model is $\Sigma^p_{k+1}(A)$-complete.
2. The analogous problem for $LP_\infty(A)$ it is $PSPACE(A)$-complete.

Theorem 4.
1. The problem of deciding whether a given variable a is in a model of an $LP_k(A)$ program is Σ^p_{k+1}-complete.
2. The analogous problem for $LP_\infty(A)$ it is $PSPACE(A)$-complete.

3 Circumscription

There are several ways one could generalize circumscription to higher levels of the polynomial hierarchy. Perhaps the easiest way would be to take QBF_k formulas

and circumscribe them directly. The circumscribed models of a QBF_k formula $B(\boldsymbol{a})$ would be the minimal models of B under set-theoretic inclusion.

However, in this paper we are taking the approach that one should separate constraints, which might be possible to calculate a priori or possibly on some special device, from the computational part of the reasoning scheme. Hence, our version of constraint circumscription will consist of *circumscribed programs* which are sequences of clauses of the form $B(\boldsymbol{a}) \leftarrow: C(\boldsymbol{b})$, where B is a propositional formula and C is a QBF_k formula. In such a clause, B is called the *relational formula* and C is called the *constraint formula.* As usual the *Herbrand Universe* for such a program will consist of all the free variables which appear in the program. Given a circumscribed program P and a subset S of the Herbrand Universe U, let ν_S be the variable assignment which makes element of S true and those of $U \backslash S$ false. Then we define P_S to be the following set of relational formulas:

$$\{B \mid B \leftarrow: C \in P \wedge \bar{\nu}_S(C) = 1\}.$$

We use $\mathbb{\wedge} P_S$ to denote the conjunction of formulas in P_S. Given a circumscribed program P and a set of variables M we define $CIRC[P; M]$ to be the second-order formula:

$$\mathbb{\wedge} P_M \wedge \neg \exists m[\mathbb{\wedge} P_m \wedge m \subset M].$$

A *model* for a circumscribed program P will be a variable assignment ν_M such that $CIRC[P; M]$ is true.

Definition 2. *CC_k is the class of all* finite *constraint circumscription programs with QBF_k constraints. $CC_\infty = \cup CC_k$.*

In the simplest common form of (propositional) circumscription one considers a single propositional formula F and looks for minimal models of F with respect to set theoretic inclusion. These minimal models are called the *models of the circumscribed formula F*. It is not hard too see that given a propositional formula F, the CC_0 program $F \leftarrow:$ has the same models as the circumscribed formula F. Given a CC_0 program with clauses

$$C(\boldsymbol{a_i}) \leftarrow: D(\boldsymbol{a_i}) \text{ for } 1 \leq i \leq n$$

it is also not hard to see that this program has the same models as the circumscribed formula

$$\mathbb{\wedge}_i D(\boldsymbol{a_i}) \supset C(\boldsymbol{a_i}).$$

Thus, the theory CC_0 coincides with the simplest common form of circumscription.

Our next result is a generalization of a result of Eiter and Gottlob [4].

Theorem 5.
1. The problem $\forall CCM_k$ of determining whether a given variable is in every model of a CC_k program is Π^p_{k+2}-complete.
2. The analogous problem for $\forall CCM_\infty$ is $PSPACE$-complete.

Proof. Let P be a CC_k program. To check if x is in all models (recall that a model here is a model of circumscribed formula) of P we universally examine truth assignments ν_M. We then check if there exists a smaller truth assignment, ν_m, such that either $\bar{\nu}_M(\bigwedge P_M) = 0$ or $\bar{\nu}_m(\bigwedge P_m) = 1$ or $\nu_m(x) = 1$. If there exists M and m for which this is not the case, then x is not in all models. Hence, $\forall CCM_k$ is in Π^p_{k+2}.

To see $\forall CCM_k$ is Π^p_{k+2}-complete let F be a Π^q_{k+2} formula $\forall \boldsymbol{x} \exists \boldsymbol{y} E(\boldsymbol{x}, \boldsymbol{y})$ where $E \in \Pi^q_k$. Let $\boldsymbol{z}$, u, and w be new variables. Consider the following CC_k program:

(1) $\quad \bigwedge_i (x_i \not\equiv z_i) \leftarrow:$
(2) $\quad u \leftarrow: E(\boldsymbol{x}, \boldsymbol{y})$
(3) $\quad \neg u \leftarrow: \neg E(\boldsymbol{x}, \boldsymbol{y})$
(4) $\quad [u \vee (w \wedge \bigwedge_j y_j)] \leftarrow:$

We claim F is true if and only u is in every model of P. Clause (1) ensures that if $m_1 \subseteq m_2$ are models, then m_1 must agree with m_2 on the x_i's and z_i's and guarantees for each possible subset of the x_i's there is a model of P with exactly those x_i's. Suppose u is in every model of P. We want to show F is true. Let us assume F is false and derive a contradiction. F being false implies $\exists \boldsymbol{x} \forall \boldsymbol{y} \neg E(\boldsymbol{x}, \boldsymbol{y})$. Let $\boldsymbol{x}$ be such that $\forall \boldsymbol{y} \neg E(\boldsymbol{x}, \boldsymbol{y})$. Let M be the model containing $\boldsymbol{x}$. Then by clause (3), $\bigwedge P_M$ must have $\neg u$ as a conjunct. So u will not be in this minimal model. This contradicts the assumption that u was in every model.

For the other direction, suppose F is true. We want to show every model of P contains u. Let M be a model of P. Suppose $\nu_M(E(\boldsymbol{x}, \boldsymbol{y})) = 0$. Then $\bigwedge P_M$ would be:

$$\bigwedge_i (x_i \not\equiv z_i) \wedge \neg u \wedge [u \vee (w \wedge \bigwedge_j y_j)].$$

So w and all the y_js would have to be in M. But F being true means we could have found a smaller model since there exists a choice of y_is making E true and hence we can make w false in that model. Hence, M must have $\nu_M(E(\boldsymbol{x}, \boldsymbol{y})) = 1$. But then by (2), M must contain u.

This theorem generalizes when one adds oracle sets to the theories. Hence, the corresponding problem for $CC_k(A)$ is $\Pi^p_{k+2}(A)$-complete.

4 Default Logic

We now develop a version of default logic similar in spirit with the versions of logic programming and circumscription we have just developed.

A *quantified Boolean default theory*, or just *default theory*, is a pair $\langle D, W \rangle$ where D is a collection of default rules:

$$\frac{\alpha : B_1(\boldsymbol{b_1}), \ldots, B_m(\boldsymbol{b_m})}{\gamma}$$

and W is a set of propositional formulas. The α and γ in each rule are propositional formulas and the B_i are quantified Boolean formulas with free variables

$\boldsymbol{b_i}$. α is called the *prerequisite*, the B_i are called the *constraints*, and γ is called the consequent of the rule. The *Herbrand base*, U for a default theory is the set of free variables appearing in the default rules and W. $Cn(W)$ is the set of all formulas provable from W using propositional tautologies and modus ponens. Given a set S of propositional formulas, a default rule is said to be S-applicable if $B_i \cup S$ is consistent for each justification B_i. If the prerequisite of a default rule also happens to be in S, we say that the rule is *strongly S-applicable*. We write D_S for the set or rules

$$\{\frac{\alpha}{\gamma} | \frac{\alpha : B_1(\boldsymbol{b_1}), \ldots, B_m(\boldsymbol{b_m})}{\gamma} \in D \text{ an } S\text{-applicable rule}\}.$$

$D_{S,w}$ is the subset of D_S generated by strongly S-applicable rules. Given a set of rules X, $Cn^X(W)$ is the set of all formula which can be proved from W using propositional tautologies, modus ponens, or rules from X.

An *extension* for a default theory $\langle D, W\rangle$ is a set of formulas S such that $Cn^{D_S}(W) = S$. A *stable model* for a default theory $\langle D, W\rangle$ is a truth assignment satisfying an extension of $\langle D, W\rangle$. A *weak extension* for a default theory $\langle D, W\rangle$ is a set of formulas S such that $W \subseteq S$ and $Cn^{D_{S,w}}(S) = S$. A *supported model* for a default theory $\langle D, W\rangle$ is a satisfying truth assignment to the variables of a weak extension of $\langle D, W\rangle$. It should be noted that an inconsistent theory cannot have either type of model. When we say *model* of a default theory we mean stable model. Supported models of default theories will be discussed again in the succinctness section.

We define $B^{D_S}(T) = T \cup \{\gamma | \frac{\alpha}{\gamma} \in D_S \text{ and } \alpha \in Cn(T)\}$. To denote iterating this operation n times we write $B^{D_S} \uparrow n(T)$, and we write $B^{D_S} \uparrow \omega(T)$ to denote iterating this operation ω times. If $\langle D, W\rangle$ is a finite theory than there exists and n such that $B^{D_S} \uparrow n(W) = B^{D_S} \uparrow (n+1)(W) = B^{D_S} \uparrow \omega(W)$.

We begin with a couple of observations about default theories with quantified Boolean constraints. Their proofs are the same as in the classical default theory case.

1. A theory S is an extension for a default theory $\langle D, W\rangle$ if and only if $S = Cn(B^{D_S} \uparrow \omega(W))$.
2. If S is an extension of $\langle D, W\rangle$ then there is a subset Z of the consequences of D such that $S = Cn(W \cup Z)$.

Definition 3. *DL_k is the class of all* finite *default theories all of whose rules have QBF_k constraints. $DL_\infty = \cup DL_k$.*

Notice DL_0 is exactly the same as the usual default logic.

We begin with a generalization of the fundamental result of Gottlob [11].

Theorem 6.
1. The problem $\exists EDL_k$ of deciding whether $\langle D, W\rangle \in DL_k$ has an extension is Σ^p_{k+2}-complete.
2. The analogous problem for DL_∞ it is $PSPACE$-complete.

Proof. Let $\langle D, W\rangle \in DL_k$. To show the problem of finding an extension is in Σ^p_{k+2} we use the two observations above. We guess a subset Z of the conclusions of the rules. We take S to be $Cn(W \cup Z)$ and check that $S = Cn(B^{D_S} \uparrow \omega(W))$. First we must figure out what D_S is. It is not hard to see that a default $d = \frac{\phi : B_1(\boldsymbol{b_1}), \ldots B_m(\boldsymbol{b_m})}{\gamma}$ contributes $\frac{\phi}{\gamma}$ if and only if for each i, $\{B_i(\boldsymbol{b_i})\} \cup W \cup Z$ is consistent. Next, we check if $\{B_i(\boldsymbol{b_i})\} \cup W \cup Z$ is consistent. This requires looking for a vector $\boldsymbol{b_i}$ such that a Π^p_k property holds. Thus, we would need a single Σ^p_{k+1} oracle call. To do this for each default we would need less than $|D|$ many Σ^p_{k+1} oracle calls. Next we construct $B = B^{D_S} \uparrow \omega(W)$. This takes at most $|D_S|^2$ many co-NP calls to check if a given prerequisite is a consequence of the set generated so far. Lastly we need to check that $Cn(B) = Cn(W \cup Z)$.

Let Φ be the conjunction of formulas in B and Ψ be the conjunction of formulas in $W \cup Z$. Then checking the above condition amounts to checking if $\Phi \equiv \Psi$ is a tautology. So after our initial guess of Z, all our oracle calls are to sets contained in Σ^p_{k+1}. Thus, the entire procedure outlined above is $NP^{\Sigma^p_{k+1}} = \Sigma^p_{k+2}$.

To see $\exists EDL_k$ is Σ^p_{k+2}-complete, let $B(\boldsymbol{a}, \boldsymbol{b}) \in \Sigma^q_k$ where $\boldsymbol{a} = (a_1, \ldots, a_m)$ and $\boldsymbol{b} = (b_1, \ldots, b_n)$ are the free variables in B. Consider the following default theory $\langle D^*, W^*\rangle$:

(1) $\quad \frac{: a_i}{a_i} \qquad \frac{\neg a_i}{\neg a_i} \qquad i = 1, \ldots, m$

(2) $\quad \frac{: \neg\gamma}{a_i} \qquad \frac{: \neg\gamma}{\neg a_i} \qquad i = 1, \ldots, m$

(3) $\quad \frac{: B(\boldsymbol{a}, \boldsymbol{b})}{\gamma}$

$\langle D^*, W^*\rangle$ has an extension S implies

$$Cn(\pm a_1 \wedge \pm a_2 \wedge \ldots \pm a_m \wedge \pm\gamma) \models \gamma.$$

The $\pm a_i$ is supposed to be either a_i or $\neg a_i$ depending upon which of these literals belongs to S. From the above we thus have:

$$Cn(\pm a_1 \wedge \pm a_2 \wedge \ldots \pm a_m) \models B(\boldsymbol{a}, \boldsymbol{b}).$$

Consequently, we have

$$Cn(\pm a_1 \wedge \pm a_2 \wedge \ldots \pm a_m) \models \forall \boldsymbol{w} B(\boldsymbol{a}, \boldsymbol{w}).$$

This in turn implies that the sentence $\exists \boldsymbol{x} \forall \boldsymbol{w} B(\boldsymbol{x}, \boldsymbol{w})$ is true. We can reverse all the above implications. That is, if $\neg\exists \boldsymbol{x} \forall \boldsymbol{w} B(\boldsymbol{x}, \boldsymbol{w})$, then there is no choice of $\pm x_i$'s which satisfies $\forall \boldsymbol{w} B(\boldsymbol{a}, \boldsymbol{w})$. So γ cannot be derived. Thus $\langle D, W\rangle$ has no extensions.

So existence of an extension of default theories of the form $\langle D^*, W^*\rangle$ is equivalent to the Σ^p_{k+2}-complete problem of whether a Σ^q_{k+2} sentence is valid.

DL_∞ case is proven in a similar fashion.

Corollary 1.
1. The problem of deciding whether $\langle D, W\rangle \in DL_k$ has a model is Σ^p_{k+2}-complete.
2. The analogous problem for DL_∞ it is $PSPACE$-complete.

Proof. To see the problem is in Σ^p_{k+2} let $\langle D, W\rangle \in DL_k$. We now slightly modify the procedure to find an extension of a default theory used in Theorem 6. At the same time, as we guess a subset Z of the conclusion we also guess a model M for $\langle D, W\rangle$. Then when we check that whether $\Phi \equiv \Psi$ is a tautology we also check if $M \models \Phi$. Only if both of these conditions are true is M a model $\langle D, W\rangle$. This procedure is still in Σ^p_{k+2}.

To see that our problem is Σ^p_{k+2}-complete notice the default theory used in Theorem 6 has models if and only if the sentence $\exists \boldsymbol{x} \forall \boldsymbol{w} B(\boldsymbol{x}, \boldsymbol{w})$ is true.

DL_∞ case is proven in a similar fashion.

Corollary 2.
1. The problem of given a formula ϕ and deciding whether or not it is in an extension of $\langle D, W\rangle \in DL_k$ is Σ^p_{k+2}-complete.
2. The analogous problem for DL_∞ it is $PSPACE$-complete.

Proof. The problem is in Σ^p_{k+2} since we can use the previous corollary to guess a model M and then check if $M \models \phi$. To see that it is Σ^p_{k+2}-complete notice that the problem of whether γ is in an extension of $\langle D^*, W^*\rangle$ in Theorem 6 is equivalent to whether a Σ^q_{k+2}-formula is satisfiable.

DL_∞ case is proven in a similar fashion.

The theorem and the corollaries above generalize when one adds oracle sets to the theories DL_k. Hence, the corresponding problems for $DL_k(A)$ are $\Sigma^p_{k+2}(A)$-complete.

5 Succinctness

In [8] a notion of representation succinctness was developed as a means of comparing the relative strength of various reasoning formalisms. We will now establish several succinctness results about nonmonotonic logics with quantified Boolean constraints. These results generalize those of [8].

Definition 4. *Let A and B be any reasoning formalisms. We say that A* is at least as representationally succinct as *B, written $B \leq_s A$ if the following is true: For each knowledge base ϕ_B in B there is a knowledge base ϕ_A in A such that (a) ϕ_B and ϕ_A are defined over the same* free *variables and have the same set of models, and (b) the size of ϕ_A is polynomially bounded in the size of ϕ_b. We write $A \not\leq_s B$ if (a) and (b) fail to hold.*

It should be noted that the method of going from ϕ_A to ϕ_B need not to be effective. Next, the definition makes it is obvious that $\leq_s$ is transitive and reflexive. We can slightly weaken the definition of succinctness by replacing (a) in the above with the following:

(a') *ϕ_A contains all of ϕ_B's variables and all models of ϕ_A are expansions of models of ϕ_B.*

We will call succinctness with respect to (a') *weak succinctness* and will write $B \leq_{ws} A$ if A is at least as weakly representationally succinct as B.

It is follows from the definitions that if $B \leq_s A$ then $B \leq_{ws} A$. Moreover, (a') implies that ϕ_A can not have a model unless ϕ_B has one.

We adopt the convention that ($k \geq 0$). Let LP_k with respect to supported models will be denoted LP_k^{sup}. LP_k with respect to stable models we will just write as LP_k. We then have:

Theorem 7. *$LP_k^{sup} \leq_s LP_k$ and $LP_k \leq_{ws} LP^{sup}$. Thus, $LP_k \equiv_{ws} LP_k^{sup}$.*

Proof. Let P be an LP_k^{sup} program. A variable a is in a supported model M of P if and only if there is a clause of the form

$$a \leftarrow: p_1, \ldots, p_n : B_1, \ldots, B_m$$

and $\nu_M(p_i) = 1$ for $1 \leq i \leq n$ and $\bar{\nu}(B_j) = 1$ for $1 \leq j \leq m$. Thus, we can view the premises and constraints of such a clause as a single constraint: namely, $\mathbb{\wedge}_i\, p_i \wedge \mathbb{\wedge}_j\, B_j$. The program P' one obtains by replacing the premises and constraints of clauses in P by such single constraints will still have the same supported models. But now given a set M, P'_M will consist of clauses of the form $a \leftarrow:$. So if M is a stable model any such a in P'_M would have to be in M. So a is in a stable M if and only if there is a clause in P' whose single constraint is satisfied by M. Clearly, this is the same condition as for a being in a supported model of P'. Hence, the stable models of P' are the same as the supported models of P' and in turn the supported models of P. Consequently, $LP_k \leq_s LP_k^{sup}$.

Now take P in LP_k. Let $a_1, \ldots, a_n$ be the variables of P. Let N be the number of clauses in P. The problem of simulating P with an LP_k^{sup} program can be viewed as the problem of rewriting P's clauses without premises. Given an arbitrary subset M of the Herbrand Universe we need some way to simulate the derivation of variables when we evaluate the Horn program P_M. In evaluating P_M, we need at most n rounds before we reach a fixed point. To perform our simulation we introduce new variables $r_{i,t}$ for $1 \leq i \leq N$ and for $1 \leq t \leq N$. $r_{i,t}$ is supposed to keep track of whether rule i of P has fired by time t. Let h be the function which takes clause numbers and returns the number of the variable which is at the head of the clause. Let P' be the following LP_k^{sup} program:
For each clause in P of the form:

$a_{h(k)} \leftarrow: B$

We add to P' the clauses:
(1) $r_{h(k),t} \leftarrow: B$ where $1 \leq t \leq n$.
Next, for each clause in P of the form:

$a_{h(k)} \leftarrow a_{h(i_1)}, \ldots, a_{h(i_m)} : B$

We add to P' the clause:
(2) $r_{h(k),t+1} \leftarrow: B \wedge (\mathbb{\vee}_{h(j)=h(i_1)}\, r_{j,t}) \wedge \ldots \wedge (\mathbb{\vee}_{h(j)=h(i_m)}\, r_{j,t})$ where $1 \leq t \leq n-1$.
Finally, we add to P' the clauses:

(3) $a_{h(i)} \leftarrow: r_{i,n}$ for $1 \leq i \leq N$

Suppose M' is a model of P' (since the clauses of P' have no premises, supported and stable models of P' will be the same). Let M be M' restricted to the variables of P. Clauses in the Horn program P_M inherit a numbering from the clauses they came from in P. Hence, clauses (1) and (2) of P' will guarantee that $r_{i,t}$ is valid in M' if and only if the i-th rule of P_M could fire at time t in the evaluation of P_M. So if $r_{i,n}$ is true then $a_{h(i)}$ could be derived in P_M. Thus, rule (3) assures us that a variable is in M if and only if it could be derived in evaluating P_M. So M will be a model of the Horn program P_M and hence a stable model of P.

Suppose M is a model P. We can extend this to a model of P' by assigning r_{it} true if rule i has fired in P_M by time t, and by assigning r_{it} false, otherwise. This completes the argument.

As before, let DL_k with respect to supported models be denoted DL_k^{sup}, and DL_k with respect to stable models we will just denoted by DL_k. One can easily modify the argument given above to porove the following.

Theorem 8. *$DL_k^{sup} \leq_s DL_k$ and $DL_k \leq_{ws} DL_k^{sup}$. Thus, $DL_k \equiv_{ws} DL_k^{sup}$.*

We will show below that $CC_k \not\leq_s LP_k$. We know $LP_k^{sup} \not\leq_s CC_k$ since models of CC_k programs are pairwise incomparable as sets, yet supported models of an LP_k program can be subsets of each other. However, if we slightly modify the definition of LP_k^{sup} to only consider programs whose supported models are pairwise disjoint then CC_k is as representationally succinct as the resulting theory. Call this new theory LP_k^*. We can convert any LP_k^{sup} program P over the variables the $p_1, \ldots p_n$ into an LP_k^* program P' over the variables $p_1, \ldots, p_n, p'_1, \ldots, p'_n$. To make P' we add to P the clauses $p_i \leftarrow: \neg p'_i$ and $p'_i \leftarrow: \neg p_i$ where $1 \leq i \leq n$. It is easy to see that P' has the same number of models as P and every model in P can be extended to a model of P'. Further, each of P''s models are pairwise incomparable. Thus, LP_k^* is a natural modification of LP_k^{sup} since this construction shows $LP_k^{sup} \equiv_{ws} LP_k^*$. Hence, by Theorem 7 it also shows that $LP_k^* \equiv_{ws} LP_k$.

Theorem 9.
1. $LP_k^* \leq_s CC_k$ but $CC_k \not\leq_s LP_k$ unless $\Sigma_{k+1}^p \subseteq \Delta_{k+1}^p/poly$.
2. If K is a reasoning formalism for which model checking can be done in Δ_{k+1}^p then $CC_k \not\leq_s K$ unless $\Sigma_{k+1}^p \subseteq \Delta_{k+1}^p/poly$.

Proof. To see $LP_k^* \leq_s CC_k$ let $P \in LP_k$. By conjoining constraint formulas we can assume that the clauses in P are of the form:

$$a_i \leftarrow p_{i1}, \ldots p_{in} : B_i.$$

Consider the circumscribed program P' made up of clauses of the form:

$$a_i \leftarrow: \mathbin{\text{⩓}}_j p_{ij} \wedge B_i.$$

Suppose M is a supported model of P. For each tail of a clause that M satisfies in P, M will satisfy the corresponding tail in P'. Now if M satisfies the tail of a clause P it must also satisfy the head. P'_M is the conjunction of these heads and hence will be satisfied. As models for P are pairwise disjoint there is no $m \subset M$ such that m is a model of P_m. Hence, there is no m such m satisfies $\bigwedge P'_m$. On the other hand, suppose M' is a model of P'. If the tail of a clause in P' is satisfied by M', the head must also be satisfied since M' satisfies $P_{M'}$. Thus, the corresponding clause in P will also be satisfied. If tail of a clause is false in P' its corresponding clause will be false in P as well. Thus, M' is a model of P. As P was an arbitrary program in LP^*_k, we have shown $LP^*_k \leq_s CC_k$.

To see $CC_k \not\leq_s LP_k$ it suffices to show the second statement. That is, $CC_k \not\leq_s K$ for any reasoning formalism for which model checking can be done in Δ^p_{k+1}. Suppose $F = \exists \boldsymbol{x} E(\boldsymbol{x})$ is a Σ^q_{k+1}-sentence. Then we can convert F to a prenex normal form Σ^q_{k+1}-sentence whose matrix is in 3CNF. Our first step in this proof is to come up with a single formula which will allow us to code up sentences of this type. Our formula will use the variables $x_{01}, \ldots, x_{0n}, \ldots, x_{k1}, \ldots, x_{kn}$. There are $8 \cdot \binom{(k+1)n}{3}$ possible 3CNF clauses over these variables. For each possibility our formula will have a variable c_i. We will denote the literals in clause i by l_{i1}, l_{i2}, l_{i3}. We are now ready to state our formula.

Let $G_n(\boldsymbol{c}, \boldsymbol{x_0})$ be the formula:

$$\forall x_{11} \ldots x_{1n} \exists x_{21} \ldots x_{2n} \ldots Q_{k1} \ldots x_{kn} (\bigwedge_{1 \leq i \leq 8 \cdot \binom{(k+1)n}{3}} c_i \supset l_{i1} \vee l_{i2} \vee l_{i3})$$

Given Σ^q_{k+1}-formula F, let n be the number of variables in F. By choosing the values of the c_i's appropriately we can encode F as $F' = \exists \boldsymbol{x_0} G_n(\boldsymbol{c}, \boldsymbol{x_0})$. This encoding is polynomial in the size of F.

Let $P[n]$ be the following circumscribed program:

(1) $\quad \bigwedge_i c_i \not\equiv c'_i \leftarrow:$
(2) $\quad x_{0i} \leftarrow: y$
(3) $\quad y \leftarrow: \neg G_n(\boldsymbol{c}, \boldsymbol{x_0})$.

Let m_F be the set that contains y, all the x_{0i}'s, and c_i if $l_{i1} \vee l_{i2} \vee l_{i3}$ is in the matrix of F, and c'_i otherwise. We claim F is satisfiable if and only if m_F is not a model of the program $P[n]$. Indeed, if F is modeled by M (i.e. satisfiable), then extend M by adding those c_i's and c'_i's which are in m_F. Since $\nu_M(G_n(\boldsymbol{c}, \boldsymbol{x_0})) = 1$, $P[n]_M$ will be just $\bigwedge_i c_i \not\equiv c'_i$ and M models this formula. M does not contain y and is contained in m_F. So m_F is not a model of $P[n]$.

To prove converse, assume that m_F is not a model of $P[n]$. Then there must be an $M \subset m_F$ such that $\nu_M(\bigwedge P[n]_M) = 1$. By (1) in $P[n]$, M and m_F must each contain the same c_i's and c'_i. M must not contain y otherwise by (2) we would have $M = m_F$. Thus, there must be some choice of $\boldsymbol{x_0}$ which satisfied G_n. This would in turn mean that F is satisfiable.

Suppose now that $CC_k \leq_s K$ where model checking in K is in Δ^p_{k+1}. Then for each n there is an at most polynomially larger program $P'[n]$ in K with the same models as $P[n]$. The program $P'[n]$ will be our polynomial sized advice. We can now check if a given n variable Π^q_{k+1} formula $\neg F$ is valid: we check if m_F

is a model of $P'[n]$. This can be done in Δ^p_{k+1}. Thus, $\Pi^p_{k+1} \subseteq \Delta^p_{k+1}/poly$ which in turn implies $\Sigma^p_{k+1} \subseteq \Delta^p_{k+1}/poly$ as $\Delta^p_{k+1}/poly$ is closed under complement.

We notice that both the Theorem 9 and its proof generalize one of the main results of [8].

Corollary 3. *$CC_k \not\leq_{ws} LP_k$ unless $\Pi^p_{k+1} \subseteq \Sigma^p_{k+1}/poly$.*

Proof. Let $P[n]$ be as above. Suppose $CC_k \leq_{ws} LP_k$. Let $P'[n]$ be the LP_k program corresponding to $P[n]$. To check if a given n variable Π^q_{k+1} formula $\neg F$ is valid, we check if m_F can be expanded to a model of $P'[n]$. This can be done in Σ^p_{k+1}. Thus, $\Pi^p_{k+1} \subseteq \Sigma^p_{k+1}/poly$.

Theorem 10. *$CC_k \leq_s DL_k$ but $DL_k \not\leq_s CC_k$ unless $\Sigma^p_{k+1} \subseteq \Pi^p_{k+1}/poly$.*

Proof. To see $CC_k \leq_s DL_k$ let $P \in CC_k$. Suppose P has free variables $p_1, \ldots p_m$ and it is made up of clauses of the form $A_i \leftarrow: B_i$ where $1 \leq i \leq n$. Let $V(\boldsymbol{p})$ be the formula $\curlywedge_i B_i \supset A_i$. Let W be empty and let D be

$$\left\{ d_j = \frac{: \neg p_j, V(\boldsymbol{p})}{\neg p_j} \middle| 1 \leq j \leq m. \right\}$$

Suppose M is model of P. M must satisfy $V(\boldsymbol{p})$. We can obtain an extension of $\langle D, W \rangle$ by applying those d_j's for which the $p_j \notin M$. Since M is minimal no more defaults will fire. The only model of this extension is M. On the other hand, if S is an extension of $\langle D, W \rangle$ then its only model is the one that sets the p_j's to zero only if d_j fired. Further, this model is a minimal model of the formula $V(\boldsymbol{p})$ and hence a minimal model of P.

To see $DL_k \not\leq_s CC_k$ unless $\Sigma^p_{k+1} \subseteq \Pi^p_{k+1}/poly$, we consider a special type of Σ^q_{k+1}-sentence. We call a Σ^q_{k+1}-formula *pure* if it is in prenex normal form, its matrix is in 3CNF and, for each clause, either all the literals are positive or all the literals are negative. Satisfiability for a pure *propositional* formulas is NP-complete [7]. It follows that deciding whether a given pure Σ^q_{k+1}-sentence is valid is Σ^p_{k+1}-complete.

We are now ready to define the default theory we will need in our succinctness result. We will use the following formula $P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \boldsymbol{c'})$ to allow us to code up pure Σ^q_{k+1}-sentences involving n variables:

$$\forall x_{11} \ldots x_{1n} \exists x_{21} \ldots x_{2n} \cdots Q_{k1} \ldots x_{kn} [\curlywedge_{1 \leq i \leq \binom{(k+1)n}{3}} (c_i \supset l_{i_1} \vee l_{i_2} \vee l_{i_3})$$

$$\wedge \curlywedge_{1 \leq i \leq \binom{(k+1)n}{3}} (c'_i \supset l'_{i_1} \vee l'_{i_2} \vee l'_{i_3})]$$

A literal l_{j_v} is either a variable x_{it} or a formula X_i. A literal l'_{j_v} is either a literal of the form $\neg x_{it}$ or a formula Y_i. The three literals in the disjuncts are supposed to run over all possible ways one could pick $l_{j_1}, l_{j_2}, l_{j_3}$ or $l'_{j_1}, l'_{j_2}, l'_{j_3}$ from their corresponding set of variables. The variables c_i and c'_i are supposed to control if a given disjunct of three literals is turned on. We define the formulas X_i and Y_i below. X_i is the formula:

$$\neg b \vee \neg z_1 \vee \cdots \vee \neg z_{i-1} \vee z_i \vee \neg z_{i+1} \vee \cdots \vee \neg z_n$$

Y_i is the formula:

$$b \vee \neg z_1 \vee \cdots \vee \neg z_{i-1} \vee z_i \vee \neg z_{i+1} \vee \cdots \vee \neg z_n$$

Let $\Delta[n] = \langle D, W \rangle$ be the default theory with W defined by:

$$W = \Vee_{i=1}^{n} \neg X_i \vee \Vee_{i=1}^{n} \neg Y_i \vee (b \wedge \bigwedge_{i=1}^{n} z_i)$$

and D consisting of the following defaults:

$$(1)\ \frac{: P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \boldsymbol{c'}) \wedge c_i}{c_i} \qquad \frac{: P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \boldsymbol{c'}) \wedge c'_i}{c'_i} \quad \text{for} \quad 1 \leq i \leq \binom{n}{3}$$

$$(2)\ \frac{: \neg c_i}{\neg c_i} \qquad \frac{: \neg c'_i}{\neg c'_i} \quad \text{for} \quad 1 \leq i \leq \binom{n}{3}$$

$$(3)\ \frac{: \neg X_i}{Y_i} \qquad \frac{: \neg Y_i}{X_i} \quad \text{for} \quad 1 \leq i \leq n.$$

Notice that $\langle D, W \rangle$ is polynomial (in fact cubic) in n.

Let $F = \exists \boldsymbol{x_0} E(\boldsymbol{x_0})$ be a pure Σ^q_{k+1}-sentence over n variables. We can view F as being over some subset of the variables in x_{ij} where $0 \leq i \leq k$ and $1 \leq j \leq n$. To each clause in its matrix we can find a corresponding clause in $P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \boldsymbol{c'})$. To find the corresponding clause to a clause in F containing a literal x_{0j} or $\neg x_{0j}$ we view x_{0j} as Y_j and $\neg x_{0j}$ as X_j. Let m_F be the subset of the Herbrand base of $\Delta[n]$ which contains b and $z_1 \ldots z_n$, and contains c_i, c'_i if the corresponding clause appears in F. We claim m_F is a model of an extension of $\Delta[n]$ if and only if F is valid. Suppose F is valid and let $x_{0i_1} \ldots x_{0i_m}$ be a vector which satisfies E. Let Z contain c_i if $c_i \in m_F$ and $\neg c_i$ if $c_i \notin m_F$. Likewise, let Z contain c'_i if $c'_i \in m_F$ and $\neg c'_i$ if $c'_i \notin m_F$. For each $1 \leq i \leq n$ let Z contain Y_i if i is an 'on' bit in the assignment satisfying E and if not let Z contain X_i. $Cn(Z \cup W)$ will be a an extension. Since F was valid, $P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{c}, \boldsymbol{c'})$ is consistent with W and Z. Thus, rules (1) and (2) will let us derive back precisely the c_i, c'_i, $\neg c_j$, and $\neg c'_j$'s in Z. Next, the rule (3) forces us to derive back the same X_i's and Y_i's. It is easy to see that m_F is a model of $Cn(Z \cup W)$. On the other hand, if m_F is a model of $\Delta[n]$, we need to show F is valid. Let S be the extension that m_F models. S must have exactly those c_i's and c'_i's needed to code F into P. This follows from the definition of m_F and the fact that m_F models S. Now in order for D_S to be able to derive back these c_i's and c'_i's there must be some choice of the X_i's and Y_i's such that P is consistent with S. The rules (3) imply that only one of X_i and Y_i can hold for each i. We can thus assign x_{0i} true if Y_i holds and x_{0i} false otherwise. Since P was consistent with S and as F was encoded with the c_i's and c'_i into P this assignment must satisfy E. Hence, F is valid as E satisfiable.

Suppose $DL_k \leq_s CC_k$. Then for each n there is circumscribed program $D[n]$ with the same models as the theory $\Delta[n]$ and polynomial in the size of $\Delta[n]$. These $D[n]$ will be our polynomial sized advice. To decide if a pure Σ^q_{k+1}-sentence on n variables is valid one need only check if m_F is a model of $D[n]$. This involves checking that no subset M of m_F satisfies $\bigwedge D[n]_M$, which can be done in Π^p_{k+1}. Thus, we reduced a Σ^p_{k+1} problem to a $\Pi^p_{k+1}/poly$ one.

Theorem 11. $DL_k \leq_{ws} LP_{k+1}$.

Proof. Consider a DL_k theory $\langle D, W\rangle$ using variables $x_1, \ldots, x_n$. By Theorem 8 it suffices to consider the case where defaults in $\langle D, W\rangle$ are prerequisite-free. Our first task is to come up with a formula $Model_{\langle D,W\rangle}(\boldsymbol{x})$ which will be true if and only if a truth assignment to $\boldsymbol{x}$ is a model of $\langle D, W\rangle$. By conjoining constraint formulas together we can assume defaults in $\langle D, W\rangle$ to be of the form:

$$\frac{: C_i(\boldsymbol{x})}{\gamma_i(\boldsymbol{x})}.$$

Let $ZW(\boldsymbol{x}, \boldsymbol{y})$ be the following formula:

$$\mathbb{M}_i\ (\gamma_i(\boldsymbol{x}) \supset (W(\boldsymbol{y}) \wedge \gamma_i(\boldsymbol{y})).$$

ZW will play the role that $Z \cup W$ plays in extension existence. In extension existence, however, we need to guess a subset of the conclusion of rules. In the present case, we will be given a model, so it will be only necessary to see which conclusions the model satisfies. We now introduce a formula which expresses that the ith rule of D contributes to D_S. Let $D_{S,i}(\boldsymbol{x})$ be the formula:

$$\forall \boldsymbol{y} C_i(\boldsymbol{y}) \wedge ZW(\boldsymbol{y}, \boldsymbol{x}).$$

Using ZW and $D_{S,i}$ we can now express $Model_{\langle D,W\rangle}(\boldsymbol{x})$ as the formula:

$$ZW(\boldsymbol{x}, \boldsymbol{x}) \wedge \forall \boldsymbol{z}[(D_{S,i}(\boldsymbol{x}) \supset \gamma_i)) \equiv ZW(\boldsymbol{z}, \boldsymbol{x})]$$

This is a QBF_{k+1} formula so it can be used in LP_{k+1} programs. The following short LP_{k+1} program P has the same models as $\langle D, W\rangle$:

(1) $\quad x_i \leftarrow: Model_{\langle D,W\rangle}(x_1, \ldots, x_n) \wedge x_i \qquad i = 1, \ldots, n$

(2) $\quad x_1 \leftarrow: (\neg Model_{\langle D,W\rangle}(x_1, \ldots, x_n)) \wedge \mathbb{M}_{1 \leq i \leq n}\ (\neg x_i).$

Let ν be a truth assignment. Suppose ν is model of $\langle D, W\rangle$. Then clauses (1) permit derivations of all those x_i's for which $\nu(x_i) = 1$. Moreover, no other x_i's can be derived. On the other hand, if we have a truth assignment ν to the x_i's which is not a model of $\langle D, W\rangle$ then there are two cases. Either none of the x_i's with $\nu(x_i) = 1$ will be derivable from P or ν has all the x_i's set to zero in which case x_1 can be derived from P. In either case, ν is not a model of P.

Thus, the models of P are precisely the models of $\langle D, W\rangle$. Notice that the clauses in P have no premises so stable and supported models of P will coincide. So we have shown both $DL_k \leq_{ws} LP_{k+1}$ and $DL_k \leq_{ws} LP^{sup}_{k+1}$.

Let $Model_{\phi_K}(\boldsymbol{x})$ be the predicate which is true if and only if $\boldsymbol{x}$ is a model of the knowledge base ϕ_K in the reasoning formalism K. Since satisfaction of a QBF_k formula is a Σ^p_{k+1}-complete problem, it seems likely that we can express $Model_{\phi_K}(\boldsymbol{x})$ as a QBF_k formulas for many formalisms for which model checking is in Δ^p_{k+1}. Then using essentially the same program as in Theorem 11 we can show:

Corollary 4. *Suppose K is a propositional reasoning formalism for which the predicate $Model_{\phi_K}(\boldsymbol{x})$ can be expressed as a QBF_k formula. Then $K \leq_{ws} LP_k$.*

This gives some evidence that LP_k, CC_k, and DL_k are nice hierarchies in which to study the succinctness of reasoning formalisms for which model checking is in the polynomial hierarchy.

Theorem 12. *$LP_k \leq_s DL_k$ but $DL_k \not\leq_s LP_k$ unless $\Sigma^p_{k+1} \subseteq \Pi^p_{k+1}/poly$.*

Proof. The second part of the theorem follows from Theorem 9 and Theorem 10. To prove the first part, let $P \in LP_k$. We translate P into the default theory $\langle D, \emptyset \rangle$ where clauses of the form

$$p \leftarrow a_1, \ldots, a_n : B_1(\boldsymbol{b}_1), \ldots, B_n(\boldsymbol{b}_m)$$

become defaults:

$$\frac{a_1 \ldots, a_n : B_1(\boldsymbol{b}_1), \ldots, B_n(\boldsymbol{b}_m)}{p}.$$

It is easy to see P and $\langle D, W \rangle$ have the same models.

Theorem 13. *$LP_{k+1} \leq_{ws} DL_k$.*

Proof. Given an LP_{k+1} program P with QBF_{k+1} constraints we first want to convert it to an LP_{k+1} program with all constraint formulas Σ^q_{k+1} or Π^q_{k+1}. By conjoining the constraint formulas we can assume each clause has only one constraint formula. By pushing negations inward we can assume each constraint formula involves only AND's and OR's of Σ^q_{k+1} and Π^q_{k+1} formulas. We eliminate propositional connectives one at a time for these QBF_{k+1} constraint formulas by repeatedly using one of the following transformations. A clause of the form:

$$a \leftarrow p_1, \ldots, p_n : B \wedge C$$

where $B \wedge C \notin \Sigma^q_{k+1}$ becomes

$$\begin{array}{l} b_1 \leftarrow p_1, \ldots, p_n : B \wedge b_2 \\ b_2 \leftarrow p_1, \ldots, p_n : b_1 \wedge C \\ a \leftarrow b_1 \wedge b_2 : \end{array}$$

where b_i are new variables not appearing in P. Notice in any model of the resulting program either a, b_1 and b_2 are true or they are all false. Thus, so far the transformation is a weakly succinct one. A clause of the form:

$$a \leftarrow p_1, \ldots, p_n : B \vee C$$

where $B \vee C \notin \Sigma^q_{k+1}$ becomes

$$\begin{array}{l} a \leftarrow p_1, \ldots, p_n : B \\ a \leftarrow p_1, \ldots, p_n : C. \end{array}$$

This type transformation adds no variables and does not change the models of P. Hence, applying both these types of transformations to clauses in P will yield a program P' and the total transformation is a weakly succinct one. Once every constraint is either Σ^q_{k+1} or Π^q_{k+1}, we convert the resulting logic program P' into a default theory $\langle D, W\rangle$ where $W = \emptyset$. Each clause in P' of the form:

$$a \leftarrow p_1, \ldots, p_n : A$$

is converted into a default of the form:

$$\frac{\lll p_i : A}{a}.$$

Given a model M of P' it is obvious that $Cn(M)$ will be an extension of $\langle D, W\rangle$, so M will be a model of $\langle D, W\rangle$. Conversely, the variables which appear in an extension of $\langle D, W\rangle$ will give a model of P'.

To make $\langle D, W\rangle$ a DL_k theory we replace each Σ^q_{k+1} constraint formula $A = \exists x_1, \ldots x_n B(\boldsymbol{x}, \boldsymbol{c})$ by a new variable γ_A. We then add defaults of the form:

$$\frac{: b_i}{b_i} \qquad \frac{: \neg b_i}{\neg b_i} \qquad \text{for} 1 \leq i \leq n \qquad (**)$$

and

$$\frac{: B(\boldsymbol{b}, \boldsymbol{c})}{\gamma_A}$$

where b_i are new variables. The defaults (**) guarantee any extension of the resulting theory will have some choice of the b_i's. Thus, γ_A will be in any such extension if and only if $\exists \boldsymbol{b} B(\boldsymbol{b}, \boldsymbol{c})$.

We replace each Π^q_{k+1} constraint formula $A = \forall x_1, \ldots, x_n B(\boldsymbol{x}, \boldsymbol{c})$ by a new variable γ_A and add a default:

$$\frac{: B(\boldsymbol{b}, \boldsymbol{c})}{\gamma_A}$$

where b_i are new variables. When we try to find an extension S of the resulting default theory, $B(\boldsymbol{b}, \boldsymbol{c})$ will be consistent with S if and only if $\forall \boldsymbol{b} B(\boldsymbol{b}, \boldsymbol{c})$ holds. So γ_A will be if S if and only if $\forall \boldsymbol{b} B(\boldsymbol{b}, \boldsymbol{c})$ holds.

Thus, extensions of $\langle D, W\rangle$ can be made extensions of $\langle D', W'\rangle$ by adding appropriate choices of γ_A's. Eliminating such variables from an extension of $\langle D', W'\rangle$ will yield an extension of $\langle D, W\rangle$.

Consequently, $\langle D', W'\rangle$ is a weakly succinct transformation of $\langle D, W\rangle$ and so also of P. Thus, $LP_{k+1} \leq_{ws} DL_k$.

We get then the following corollary:

Corollary 5. $DL_k \equiv_{ws} LP_{k+1}$.

Using the results of Baker, Gill, Solovay [1] and Furst, Saxe, Sipser [6] we can construct an oracle A for which $\Sigma^p_k(A) \neq \Pi^p_k(A)$ for $k \geq 1$. The various conditional succinctness inequalities we have obtained above thus become strict for

the formalisms $LP_k^*(A)$, $LP_k(A)$, $CC_k(A)$ and $DL_k(A)$. Similarly, if we choose a $PSPACE$-complete oracle then as the polynomial hierarchy collapses to the first level with respect to this oracle so to do our succinctness hierarchies. In the same vein, the translations used above show that $LP_\infty \equiv_{ws} CC_\infty \equiv_{ws} DL_\infty$.

We now spend a moment to mention the results of the interesting paper of Eiter, Lu, and Subrahmanian [5] which also appears in this volume. Their setting is predicate logic programming. They develop a notion of constrained interpretation and constrained logic program to avoid having to define the stable models of logic programs in terms of minimal Herbrand models of their ground instantiations which may, in general, be infinite. In the function free case their constraints turn out to be first-order formulas in the language with equality. For bounded numbers of quantifier alternations the problem of satisfaction or validity of such formulas is complete for various levels of the polynomial hierarchy and so in turn problems concerning constrained logic programs are complete for various levels of the hierarchy. It might be possible to interpret their results into a reasoning formalism with quantified Boolean constraints since if we have an equality $X = Y$ where X,Y can range over finitely many values we can do a propositional translation of this equality as $\wedge_i(x_i \Leftrightarrow y_i)$ where x_i and y_i represent the bits in the coding of X and Y. Such translations might be of practical importance in a hardware implementation of their scheme and warrant further study.

References

1. T. Baker, J. Gill, and R. Solovay ."Relativizations of the P =? NP question ". *SIAM Journal of Computing.* , 1993. 431-442.
2. M. Cadoli and M. Schaerf. 'A survey on complexity results for non-monotonic logic" *Journal of Logic Progrmming.* , Nov. 1993, vol.17,(no.2-4):127-60.
3. W.F. Dowling and J.H. Gallier. "Linear-time algorithms for testing the satisfiability of propositional Horn formulae". *Journal of Logic Programming*, 3:267–284, 1984
4. T. Eiter and G. Gottlob. "Propositional circumscription and extended closed-world reasoning are Π_2-complete". *Theoretical Computer Science*, 114:231-245, 1993.
5. T. Eiter and J. Lu and V.S.Subrahmanian. "Computing non-ground representations of stable models". In: *Logic Programming and Nonmonotonic Reasoning, LPNMR'97* This volume
6. M. Furst and J.B.Saxe and M. Sipser. "Parity, circuits and the polynomial hierarchy". *Math Systems Theory*, 17:13-27, 1984.
7. M. R. Garey and D. S. Johnson. Computers and Intractibility: A Guide to the theory of NP-completeness. W.H. Freeman and Company, 1979
8. G. Gogic , H. Kautz, C. Papadimitriou, and B. Selman. "The comparative linguistics of knowledge representation". In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI-95*. Springer Lecture notes in Artificial Intelligence; 1042. pages 862-869, 1995.
9. G. Gogic. Complexity Aspects of Knowledge Representation. Ph.D. Thesis. Computer Science Department, U.C. San Diego. 1996.
10. J. Goldsmith and D. Joseph. Three results on the polynomial isomorphism of complete sets. In *Foundations of Computer Science*, volume 27, 1986.

11. G. Gottlob. "Complexity results for nonmonotonic logics". *Journal of Logic and Computation*, 2:397–425, 1992
12. J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In: *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Münich, 1987.
13. J. Jaffar and M. Maher Constraint Logic Programming: A Survey. *Journal of Logic Programming* 19-20, pages 503–581. 1994.
14. V.W. Marek and M. Truszczyński. Nonmonotonic Logic. Springer Verlag. 1993.
15. V.W. Marek, A. Nerode and J.B. Remmel. "On logical constraints in Logic Programming". In: *Logic Programming and Nonmonotonic Reasoning, LPNMR'95*, Springer Lecture Notes in Artificial Intelligence 928, pages 43–56, 1995.
16. A.R. Meyer and L.J. Stockmeyer. "The equivalence problem for regular expressions with squaring requires exponential time". In *Proceedings of the 13th annual symposium on switching and automata theory*, pages 125–129. New York, NY: IEEE Computer Society, 1972.
17. C.H. Papadimitriou. "The complexity of knowledge representation". In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pages 244-248. IEEE Computer Society, 1996.
18. U. Schöning. "A note on complete sets for the polynomial hierarchy". *ACM SIGACT News* **13** pages 30–34. 1981