Answer Set Programming with Default Logic

Victor W. Marek
Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA,
marek@cs.uky.edu

Jeffrey B. Remmel
Department of Mathematics
University of California
La Jolla, CA 92093, USA
jremmel@ucsd.edu

Abstract

We develop an Answer Set Programming formalism based on Default Logic. We show that computing generating sets of extensions in this formalism captures all Σ_2^P search problems.

I. Introduction

The main motivation for this paper comes from recent developments in knowledge representation theory. In particular, a new generation of general solvers have been developed, (Niemelä and Simons 1996; Eiter et. al. 1998; Cholewiński et.al. 1999; Syrjanen 2001; Simons et. al. 2002), based on the so-called Answer Set Programming (ASP) paradigm (Niemelä 1998; Marek and Truszczyński 1999; Lifschitz 1999). The most popular ASP formalism is based on the the stable semantics for logic programs (SLP) (Gelfond and Lifschitz 1988). However, one can easily extend the ideas of answer set programming to other nonmonotonic logic formalisms such as default logic (Reiter 1980). In each case, the first question one should ask is what exactly can these systems theoretically compute. In (Marek and Remmel 2001), the authors answered this question for ASP systems built on SLP. Namely, answer set programs under SLP can solve the class of NP-search problems and no more. The main result of this paper is to prove a similar result for ASP systems built on Default Logic (DL). That is, we shall show that ASP systems built on DL can solve the class of $\Sigma_2^{\rm P}$ search problems and no more.

Default Logic has been introduced by Raymond Reiter in his seminal paper (Reiter 1980). The formalism of Default Logic has been, subsequently, extensively studied by the Knowledge Representation community. In addition to the original semantics of *extensions*, many additional structures associated with a given default theory $\langle D, W\rangle$ have been introduced. Those include weak extensions (Marek and Truszczyński 1989), Łukaszewicz extensions (Łukaszewicz 1984), rational extensions (Mikitiuk and Truszczyński 1993) and other structures. For a detailed discussion of Default Logic with extensions see (Marek and Truszczyński 1993). Default Logic with extensions forms a direct generalization of stable semantics of logic program (the latter has been introduced by Gelfond and Lifschitz in (Gelfond and Lifschitz 1988)), see (Marek and Truszczyński 1989a; Bidoit and Froidevaux 1991). Weak extensions turned out to be equivalent to Autoepistemic Logic of Moore (Moore 1985). See (Marek and Truszczyński 1993) for details. The basic complexity result for Default Logic was established by Gottlob (Gottlob 1992) (see also Stillman (Stillman 1992). Gottlob found that the decision problems associated with the Default Logic are complete for the second level of polynomial hierarchy. Specifically, the existence problem for extensions is Σ_2^P complete, the membership problem for extensions (*membership_in_some*, *membership_in_all*) are complete, respectively, for Σ_2^P and Π_2^P.

A search problem ((Garey and Johnson 1979)) $\mathcal{S}$ has two components. First, $\mathcal{S}$ specifies a set of finite instances (Garey and Johnson 1979). For example, the search problem may be to find Hamiltonian paths in a graph so that the set of instances of the problem is the set of all finite graphs. Second, for any given instance $I \in \mathcal{S}$, $\mathcal{S}$ specifies a set S_I of solutions to the search problems $\mathcal{S}$ for instance I. For example, in our Hamiltonian path problem, given a finite graph I, S_I is the set of all Hamiltonian paths of I. An algorithm solves the search problem $\mathcal{S}$ if, given any instance I of $\mathcal{S}$, the algorithm returns a solution $s \in S_I$, whenever S_I is non-empty, and returns the string "empty" otherwise.

We say that a search problem S is in $\Sigma_2^{\rm P}$ if and only if there is a polynomial time coding procedure which maps each instance in $I \in S$ to a string x_I and there is a non-deterministic polynomial time oracle Turing machine M with an oracle $X \in {\rm NP}$ such that given a coding x_I of an instance $I \in S$, the output of any terminating computation of M^X with input x_I codes a solution $s \in S_I$ and there are no terminating computations of M^X on input x_I if $S_I = \emptyset$.

The goal of this paper is first to investigate the ASP formalism based on Default Logic (DL) which has the same basic properties as SLP Answer Set Programming formalism. This formalism is closely related to both to (Niemelä and Simons 1996) and (East and Truszczyński 2001), but allows for more complex entities. By definition, extensions of default theories are always infinite since they are closed under logical consequence. This is in contrast with stable models of logic programs which are finite. Thus to have an appropriate analogue for the result that ASP logic programs capture NP search problems, we shall consider generating sets of extensions of default theories as opposed to extensions themselves. Then we will show that any $\Sigma_2^{\rm P}$ search problem can be reduced to the problem of finding generat-

ing sets for extensions of DL programs and, vice versa, the problem of finding generating sets of extensions of DL programs is itself a $\Sigma_2^{\rm P}$ search problem. That is, we shall show that for each n and each polynomial run time bound $p(x)$, there is a single ASP default theory $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p}\rangle$ that is capable of simulating any polynomial time nondeterministic Turing oracle machine with an oracle for 3-SAT on inputs of size n in the sense that given any polynomial time nondeterministic oracle Turing machine M with an oracle for 3-SAT and any input σ of size n, there is a set of formulas $edb_{M,p,\sigma}$ such that a certain class of generating sets of the extensions of $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p} \cup edb_{M,p,\sigma}\rangle$ codes accepting computations of $M^{\text{3-SAT}}$ started with input σ that terminates in $p(|\sigma|)$ or fewer steps and any such accepting computation of $M^{\text{3-SAT}}$ is coded by the generating set of some extension of $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p} \cup edb_{M,p,\sigma}\rangle$.

Our results here are closely related to the work of Cadoli, Eiter and Gottlob (Cadoli et. al. 1994; Cadoli et. al. 1997) who studied the use of Default Logic as a query language. Recall that Reiter, from the very beginning, recognized that one can treat defaults with variables. Reiter called such defaults *open* and realized that they can be viewed as reasoning patterns. That is, instantiating an open default rule creates a new propositional default rule. In (Cadoli et. al. 1994; Cadoli et. al. 1997), a DQL Input/Output query Q consists of a pair (B, D) where B is a set of first-order formulas and D is a set of open default rules, where the first order language is function-free and quantifier-free, plus a set of output relation schemata $S = \{S_1, \ldots, S_m\}$. One assumes that the set of predicate symbols occurring in the defaults of Q contain all the names of the relation schemata $\{R_1, \ldots, R_n\}$ of the database (the extensional relations) and possibly some other symbols (the intensional relations). One assumes that the output relations are intensional. The intuitive meaning of query is the following. We want to compute all tuples in the S_i relations which can be inferred under the credulous semantics. More formally, suppose W is a database instance over the set of relation schemata $\{R_1, \ldots, R_n\}$ over a finite universe $\mathcal{U}$. If R_i is an l_i-ary relation, let $R_i|W$ be the set of l_i-tuples in W belonging to R_i. Let the completion of W, $COMP(W)$ be

$$\bigcup_{i=1}^{n}\{R(a_1,\ldots,a_{l_i}) : (a_1,\ldots,a_{l_i}) \in R_i|W\} \cup \bigcup_{i=1}^{n}\{\neg R(a_1,\ldots,a_{l_i}) : (a_1,\ldots,a_{l_i}) \in \mathcal{U}^{l_i} \setminus R_i|W\}.$$

$COMP(W)$ completely describes the finite relational system $\langle \mathcal{U}, R_1, \ldots, R_n\rangle$.

Let $INST(B)$ denote the set of ground formulas that result by uniformly substituting constants from $\mathcal{U}$ for the free variables in formulas of B and, similarly, let $INST(D)$ denote the set of ground default rules that result by uniformly substituting constants from $\mathcal{U}$ for the free variables in formulas of the open defaults in D. Let $Q + W$ denote the default theory with defaults $INST(D)$ and first-order formulas $COMP(W) + INST(B)$. Then the answer to the DLQ query $Q = \langle (B, D), \{S_1, \ldots, S_m\}\rangle$ equals

$$\bigcup_{i=1}^{m}\{S_i|W\}$$

where $S_i|W$ is the set of all ground l_i-tuples $\mathbf{t}$ over $\mathcal{U}$ such that $S_i(\mathbf{t})$ is in at least one extension of $Q + W$. It is clear that this is analogous to the way that DATALOG and DATALOG$^{\neg}$ (see (Ullman 1988)) treats queries to databases. Then the main result of (Cadoli et. al. 1994; Cadoli et. al. 1997) is that a database query is Σ_2^p-recognizable if and only if it is definable as DQL I/0 query.

The outline of this paper is as follows. In section II, we shall describe specifying our formulation of an Answer Set Programming based on default logic. We shall also formally describe our conventions for how a non-deterministic oracle machine relative to an oracle for 3-SAT operates. Then in section III, we shall describe a uniform coding our uniform coding of nondeterministic Turing machines with an oracle for 3-SAT via our ASP default theories which is used to derive our main result that our ASP default theories capture all Σ_2^p-search problems.

II. Technical preliminaries

In this section, we formally introduce several notions that will be needed for the proof of our main results. First, we shall make a typographical departure from the original convention of Reiter for writing defaults. Recall, that in Reiter's original paper and most of the literature on Default Logic, a default is written as $\frac{\alpha:\beta_1,\ldots,\beta_k}{\gamma}$. This is convenient for theoretical considerations, but it is not typographically suitable when either α and/or some of the β's are long. Gelfond and Lifschitz (Gelfond and Lifschitz 1991) suggested that default logic should be treated as a natural extension of Logic Programming. We will follow this suggestion and write a default as a rule:

$$\gamma \leftarrow \alpha : \beta_1, \ldots, \beta_k.$$

First, we introduce the set of propositional default logic programs that we will study. Let $Form(\mathcal{L})$ be the set of formulas for a propositional language $\mathcal{L}$. Given $T \subseteq Form(\mathcal{L})$, we let the theory of T, $Th(T)$, denote the set of all logical consequences of T. We say that T is theory if $T = Th(T)$. We let $\mathcal{P}(Form(\mathcal{L}))$ denote the set of all subsets of $Form(\mathcal{L})$. A propositional default theory $\langle D, W\rangle$ is a pair where D is a collection of default rules and W is a subset of $Form(\mathcal{L})$. To each such default theory $\langle D, W\rangle$, we associate an operator, $\Gamma_{\langle D,W\rangle} : \mathcal{P}(Form(\mathcal{L})) \rightarrow \mathcal{P}(Form(\mathcal{L}))$, called Reiter's operator, by defining $\Gamma_{\langle D,W\rangle}(S) = T$ if T is the least theory contained in $Form(\mathcal{L})$ such that (i) $W \subseteq T$ and (ii) T satisfies the following condition: Whenever $\psi \leftarrow \alpha : \beta_1, \ldots, \beta_m \in D$, $\alpha \in T, \neg\beta_1 \notin S, \ldots, \neg\beta_m \notin S$, then $\psi \in T$. A theory $S \subseteq Form(\mathcal{L})$ is called a *default extension of* $\langle D, W\rangle$ if $\Gamma_{\langle D,W\rangle}(S) = S$. We say that G is a generating set for $\langle D, W\rangle$ if $W \subseteq G \subseteq W \cup \{\phi : \phi$ is the head of some rule $r \in D\}$ and $Th(G)$ is an extension of $\langle D, W\rangle$.

In the spirit of answer set programming, we shall extend the notion of propositional default theories to predicate logic default theories where there are *no function* symbols in the underlying predicate logic. These predicate logic default theories are the analogue of $DATALOG^{\neg}$ program used in (Marek and Remmel 2001) or PS+ theories used in (East and Truszczyński 2001). That is, we consider a predicate logic $\mathcal{L}$ where we allow predicate symbols of any arity but no function symbols. In particular, we allow predicate symbols of arity 0 which are propositional letters. The only terms of the language are either constant symbols or individual variables. In particular both Herbrand universe and Herbrand base of $\mathcal{L}$ are finite, since we will deal with finite default theories. We let $Form(\mathcal{L})$ denote the set all formulas of $\mathcal{L}$ and we let $Sent(\mathcal{L})$ denote the set of sentences of $\mathcal{L}$, i.e. the set of all formulas of $\mathcal{L}$ with no free variables. We let $QFForm(\mathcal{L})$ denote the set all quantifier free formulas of $Form(\mathcal{L})$ and $QFSent(\mathcal{L})$ denote the set of all quantifier free sentences of $Sent(\mathcal{L})$. If $\overline{X} = (x_1, \ldots, x_n)$ is a sequence of individual variables, then for any given formula $\varphi \in Form(\mathcal{L})$, we shall write $\varphi(\overline{X})$ to indicate that the free variables of φ are among the variables in $\overline{X}$.

An *ASP default theory* is a pair $\langle D, W\rangle$ where D is a finite collection of default rules, that is, rules of form

$$r(\overline{X}) = \psi(\overline{X}) \leftarrow \alpha(\overline{X}) : \beta_1(\overline{X}), \ldots, \beta_m(\overline{X}), \quad (1)$$

where $\alpha(\overline{X}), \beta_1(\overline{X}), \ldots, \beta_m(\overline{X})$, and $\psi(\overline{X})$ are quantifier free formulas in $\mathcal{L}$ and W a finite subset of $QFForm(\mathcal{L})$. Let $c_1, \ldots, c_k$ be the set of all constants that occur in $\langle D, W\rangle$. Suppose that $\overline{X} = (X_1, \ldots, X_n)$. Then ground instance of default rule $r(\overline{X})$ as in (1) is the result of a simultaneous substitution of constants $\overline{d} = (d_1, \ldots, d_n)$, where $d_i \in \{c_1, \ldots, c_k\}$ for all i, for the variables $\overline{X}$ occurring in $r(\overline{X})$. Similarly a ground instance of a formula $\varphi(\overline{X}) \in W$ is the result of a simultaneous substitution of constants $\overline{d}$ for the variables $\overline{X}$ occurring in φ. Given a ASP default theory $\langle D, W\rangle$, $\langle D_g, W_g\rangle$ is a *propositional* default theory where D_g is the set of all ground instances of rules in D and W_g of all ground instances of formulas in W. If no constant symbol occurs in $\langle D, W\rangle$, then we fix some new constant symbol c_1 and let $\langle D_g, W_g\rangle$ be the result of substituting c_1 for every variable that occurs in $\langle D, W\rangle$ so that once again $\langle D_g, W_g\rangle$ can be considered a propositional default theory. We then say that E is an extension of $\langle D, W\rangle$ if and only if E is an extension of $\langle D_g, W_g\rangle$.

Our first result is that the problem of computing generating sets an ASP default theory is a Σ_2^P-search problem. That is, fix some set of variables $\mathcal{X} = \{X_1, \ldots, X_k\}$. Then we consider the set $DF(\mathcal{X})$ of all finite ASP predicate logic default theories $\langle D, W\rangle$ whose underlying set of variables is contained in $\mathcal{X}$. Then we can define a search problem $S(\mathcal{X})$ by saying that an instance I of $S(\mathcal{X})$ is a default theory $\langle D, W\rangle$ in $DF(\mathcal{X})$ and the set of solutions of I is the set of generating sets of $\langle D_g, W_g\rangle$. It is then easy to prove the following.

Theorem 1 *For any set of variables $\mathcal{X} = \{X_1, \ldots, X_k\}$, $S(\mathcal{X})$ is a Σ_2^P search problem.*

A nondeterministic Oracle Turing Machine is a 8-tuple of the form

$$M = (Q, \Sigma, \Gamma, D, \delta, s_0, , s_q, f).$$

Here Q is a finite set of states and Σ is a finite alphabet of input symbols. We assume Q always contains three special states, s_0, the start state, s_q, the query state, and f, the final state. We also assume that there is a special symbol B for "blank" such that $B \notin \Sigma$ and $\Gamma = \Sigma \cup \{B\}$ is the set of tape symbols. The set D is the set of move directions consisting of the elements l, r, and λ where l is the "move left" symbol, r is the "move right" symbol and λ is the "stay put" symbol. The function δ is the nondeterministic transition function of the machine M.

We assume M operates on *two* one-way infinite tapes, a computation tape and a query tape, where the cells of the tapes are labeled from left to right by $0, 1, 2, \ldots$. To visualize the behavior of the machine M, we shall talk about the *two* read-write heads of the machine, the *c-read-write* head on the computation tape and the *q-read-write* head on the query tape. At any given time in a computation, the read-write heads of M are always in some state $s \in Q$ and the c-read-write head is reading some symbol $p_a \in \Gamma$ which is in some cell a on the computation tape and the q-read-write head is reading some symbol $p_b \in \Gamma$ which is in some cell b on the query tape. If $s \neq s_q$, then M picks an instruction $(s1, p1, d1, p2, d2) \in \delta(s, p_a, p_b)$ and then replaces the symbol p_a on the computation tape by $p1$, replaces the symbol p_b on the query tape by $p2$, changes its state to state $s1$, and moves on the computation tape according to d_1 and on the query tape according to d_2. If $s = s_q$, then M takes one of two actions depending on the current state of the query tape and the oracle $\mathcal{O}$. That is, if the string of symbols consisting of all cells that are weakly to the left of the right-most non-blank symbol on the query tape is in $\mathcal{O}$ (not in $\mathcal{O}$), then M picks an instruction $(s1, p1, d1, p2, d2)$ such that $\{yes\} \times (s1, p1, d1, p2, d2) \in \delta(s_q, p_a, p_b)$ $(\{no\} \times (s1, p1, d1, p2, d2) \in \delta(s_q, p_a, p_b))$ and then replaces the symbol p_a on the computation tape by $p1$, replaces the symbol p_b on the query tape by $p2$, changes its state to state $s1$, and moves on the computation tape according to d_1 and on the query tape according to d_2.

We assume that at the start of the computation of M on input σ of length n, the cells $0, \ldots, n-1$ of the computation tape contain the symbols $\sigma(0), \ldots, \sigma(n-1)$ respectively and all cells to the right of cell $n-1$ are blank. We also assume that all the symbols on the query tape are blank. We do not impose (as it is often done) any special restrictions on the state of the tape and the position of the read-write heads at the end of computation. However, we assume that at the start of any computation, the read-write heads are in state s_0 and the c-read-write head is reading the symbol in cell 0 on the computation tape and the q-read-write head is reading the symbol in cell 0 of the query tape.

Suppose we are given a oracle Turing machine M with oracle $\mathcal{O}$ whose runtimes are bounded by a polynomial $p(x) = a_0 + a_1 x + \cdots + a_k x^k$ where each $a_i \in N = \{0, 1, 2, \ldots\}$ and $a_k \neq 0$. That is, on any input of size n, an accepting computation terminates in at most $p(n)$ steps. Then any accepting computation on input σ can affect at most the first $p(n)$ cells of the both the computation and the query tapes.

Thus in such a situation, there is no loss in only considering tapes of length $p(n)$. Hence in what follows, one shall implicitly assume that the both the computation tape and the query tapes are finite. Moreover, it will be convenient to modify the standard operation of M in the following ways.
1. We shall assume $\delta(f, a, b) = \{(f, a, \lambda, b, \lambda)\}$ for all $a, b \in \Gamma$.
2. Given an input x of length n, instead of immediately halting when we first get to the final state f reading a symbol a on the computation tape and symbol b on the query tape, we just keep executing the instruction $(f, a, \lambda, b, \lambda)$ until we have completed $p(n)$ steps. That is, we remain in state f, we never move, and we never change any symbols on the tapes after we get to state f. The main effect of these modifications is that all accepting computations will run for exactly $p(n)$ steps on an input of size n.
Finally, we end this section by describing our conventions for the operation of Turing machines with an oracle for 3-SAT. First we need to discuss the coding of clauses with three literals (3-clauses, for short). A 3-*clause* is an expression of the form $\varepsilon_1 A_{i_1} \vee \varepsilon_2 A_{i_2} \vee \varepsilon_3 A_{i_3}$ where each ε_i is either empty string or $\neg$, and A_{i_j}, $j = 1, 2, 3$ are propositional atoms. If we have n propositional atoms, $A_1, \ldots, A_n$ then there is precisely $8 \cdot \binom{n}{3}$ 3-clauses based on $A_1, \ldots, A_n$. We order the 3-clauses so that the 3-clauses based on $A_1, \ldots, A_n$ form an initial segment of the 3-clauses based on $A_1, \ldots, A_{n+1}$. Thus when we write φ_i, we mean the i^{th} 3-clause in this fixed ordering. With this convention, the index i such that $\varepsilon_1 A_{i_1} \vee \varepsilon_2 A_{i_2} \vee \varepsilon_3 A_{i_3} = \varphi_i$ does not depend on the number n of atoms. There are $2^{8 \cdot \binom{n}{3}}$ propositional formulas of the form $\varphi_{1_1} \wedge \cdots \wedge \varphi_{i_m}$ where $1 \leq i_1 < \ldots < i_m \leq 8 \cdot \binom{n}{3}$. These are the possible elements of 3-SAT based on the propositional atoms $A_1, \ldots, A_n$. Our convention that $\varphi_{i_1} \wedge \cdots \wedge \varphi_{i_m}$ will be coded on the query tape by a sequence of $(e_0, \ldots, e_{8 \cdot \binom{n}{3-1}})$ where $e_i = 1$ if $i + 1 \in \{i_1, \ldots, i_m\}$ and $e_i = B$ otherwise. In our coding of Turing machine via ASP default theories, we will be given a non-deterministic polynomial time oracle Turing machine with a 3-SAT oracle and we will be given a run time $p(n)$. Thus on an input of size n, the oracle machine can visit at most $p(n)$ cells on the both the computation and the query tape. By our coding of 3-clauses, any clause that contain the propositional letter A_t has index at most $8 \cdot \binom{t}{3}$. Thus any query that we make of 3-SAT certainly cannot contain an A_t where $t > p(n)$. Thus, since there are $8 \cdot \binom{p(n)}{3}$ 3-clauses based on the propositional letters $A_1, \ldots, A_{p(n)}$, we will assume that on an input of size n, the input tape has exactly $p(n)$ cells and the query tape has exactly $8 \cdot \binom{p(n)}{3}$ cells. In that case, we are coding the $2^{8 \cdot \binom{n}{3}}$ propositional formulas of the form $\varphi_{1_1} \wedge \cdots \wedge \varphi_{i_m}$ where $1 \leq i_1 < \ldots < i_m \leq 8 \cdot \binom{n}{3}$ by strings of 0's and 1's of length $8 \cdot \binom{n}{3}$. There are many such codings that are possible, but any such coding of such conjunctions of three clauses with strings of 0's and 1's still must use $2^{8 \cdot \binom{n}{3}}$ strings and hence cannot produce a significantly shorter set of codes.

III. Uniform coding of Nondeterministic Oracel Turing Machines with a 3-SAT Oracle by a Default Logic Program

In this section, we shall prove our main result that computing generating sets of extensions of DL programs captures all Σ_2^P search programs.
We define for each n and run time polynomial p, a default theory $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p} \rangle$, and for each input σ of length n and nondeterministic polynomial time oracle Turing machine M with an oracle for 3-SAT, an extensional database $edb_{M,p,\sigma}$ which can be computed in polynomial time from $M, p,$ and σ such that (a) for each accepting computation c of M on input σ, there is a generating set G_c of a unique extension E_c of $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p} \cup edb_{M,p,\sigma} \rangle$ which codes the computation c in such a way that c can be recovered in linear time from G_c and (b) for each extension E of the $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p} \cup edb_{M,p,\sigma} \rangle$, there is an accepting computation c_E of M on input σ such that $Th(G_{c_E}) = E$.
First, we need to define the underlying language of the theory $\langle D_{Trg}^{n,p}, W_{Trg}^{n,p} \rangle$. We also explain the use for each symbol. The set of predicates that will occur in our extensional database are the following: $time(X)$ for "X is a time step", $c\text{-}cell(X)$ for "X is a cell number on the computation tape", $q\text{-}cell(X)$ for "X is a cell number on the query tape", $symb(X)$ for "X is a symbol", $state(S)$ for "S is a state", $ci\_position(P)$ for "P is the initial position of the read-write head on the computation tape", $qi\_position(P)$ for "P is the initial position of the read-write head on the computation tape", $c\text{-}data(P, Q)$ for "Initially, the computation tape stores the symbol Q at the cell P", $q\text{-}data(P, Q)$ for "Initially, the query tape stores the symbol Q at the cell P", $tape_c(X, Y, T)$ for "X is symbol in cell Y on the computation tape at time T", $tape_q(X, Y, T)$ for "X is symbol in cell Y on the query tape at time T", $delta(X, Y, Z, X1, Y1, M1, Y2, M2)$ for "the 5-tuple $(X1, Y1, M1, Y2, M2)$ is an executable instruction when the read-write head is in state $X \in Q - \{s_q\}$ and is reading the symbol Y on the computation tape and the symbol Z on the query tape", $delta_{yes}(s_q, Y, Z, X1, Y1, M1, Y2, M2)$ for "the 5-tuple $(X1, Y1, M1, Y2, M2)$ is an executable instruction when the read-write head is in state s_q and is reading the symbol Y on the computation tape and the symbol Z on the query tape and the oracle gives the answer yes", $delta_{no}(s_q, Y, Z, X1, Y1, M1, Y2, M2)$ for "the 5-tuple $(X1, Y1, M1, Y2, M2)$ is an executable instruction when the read-write head is in state s_q and is reading the symbol Y on the computation tape and the symbol Z on the query tape" and the oracle gives the answer no", $neq(X, Y)$ for "X is different from Y" , $eq(X, Y)$ for "X is equal to Y", $succ(X, Y)$ for "Y is equal to $X + 1$"[1] $content(A, X, T)$ for the the predicate that A is the symbol in cell X of the query tape at time T.
Now fix a polynomial time Turing machine $M = (Q, \Sigma, \Gamma, D, \delta, s_0, s_q, f)$ with a 3-SAT oracle, an input $\sigma =$

[1]For the clarity of presentation we will use equality symbol $=$, inequality symbol, $\neq$ and relation described by the successor function $+1$, instead of eq, neq, and $succ$.

$(\sigma(0),\ldots,\sigma(n-1))$ of length n, and a run-time polynomial $p(x)$. This given, we now define the extensional database $ext_{M,p,\sigma}$. First, $ext_{M,p,\sigma}$ will contain the following the following set of constant symbols: (1) $0,1,\ldots,8\cdot\binom{p(n)}{3}$, (2) s, for each $s\in S$ (Note three constants s_0 (for initial state), s_q (for query state) and f (for final state) will be present in every extensional database), (3) B (blank symbol) and x for each $x\in\Sigma$, and (4) r,l,λ.
Our extensional database $edb_{M,\sigma,p}$ will consist of two groups. The first group of facts consists of the following set of facts that describe the machine M, i.e. the basic declarations for the predicates $state$, $symb$, and $delta$, the segment of integers $0,\ldots,8\cdot\binom{t}{3}$, i.e. the basic declarations for the predicates $time$, c-$cell$, q-$cell$, and predicates that describe the initial configuration the computation tape on input σ.

(1) For each $s\in Q$, the clause $state(s)$ belongs to $ext_{M,p,\sigma}$.
(2) For each $x\in\Gamma$, the clause $symb(x)$ belongs to $ext_{M,p,\sigma}$.
(3) For every triple $(s,x,y)\in Q-\{s_q\}\times\Gamma\times\Gamma$ and every 5-tuple $(s1,x1,d1,x2,d2)\in\delta(s,x,y)$,
the clause $delta(s,x,y,s1,x1,d1,x2,d2)$ belongs to $ext_{M,p,\sigma}$.
(4) For every pair $(x,y)\in\Gamma\times\Gamma$ and every 5-tuple $(s1,x1,d1,x2,d2)$ such that $(yes,(s1,x1,d1,x2,d2))\in\delta(s_q,x,y)$, the clause
$delta_{yes}(s_q,x,y,s1,x1,d1,x2,d2)$ belongs to $ext_{M,p,\sigma}$.
(5) For every pair $(x,y)\in\Gamma\times\Gamma$ and every 5-tuple $(s1,x1,d1,x2,d2)$ such that $(no,(s1,x1,d1,x2,d2))\in\delta(s_q,x,y)$, the clause
$delta_{no}(s_q,x,y,s1,x1,d1,x2,d2)$ belongs to $ext_{M,p,\sigma}$.
(6) For $0\leq i<8\cdot\binom{p(n)}{3}$, the clause $succ(i,i+1)$ belongs to $ext_{M,p,\sigma}$.
(7) For $0\leq i\leq p(n)$, the clause $time(i)$ belongs to $ext_{M,p,\sigma}$.
(8) For $0\leq i\leq p(n)-1$, the clause c-$cell(i)$ belongs to $ext_{M,p,\sigma}$.
(9) For $0\leq i\leq 8\cdot\binom{p(n)}{3}-1$, the clause q-$cell(i)$ belongs to $ext_{M,p,\sigma}$.
(10) For $0\leq m\leq|\sigma|-1$, the clause c-$data(m,\sigma(m))$ belongs to c-$ext_{M,p,\sigma}$.
(11) For $|\sigma|\leq m\leq p(n)-1$, the clause c-$data(m,B)$ belongs to $ext_{M,p,\sigma}$.
(12) For $0\leq m\leq 8\cdot\binom{p(n)}{3}-1$, the clause q-$data(m,B)$ belongs to $ext_{M,p,\sigma}$.
(13) The clauses $dir(l)$, $dir(r)$ and $dir(\lambda)$ belong to $ext_{M,p,\sigma}$.
(14) The clauses $ci\_position(0)$ and $qi\_position(0)$ belong to $ext_{M,p,\sigma}$.
(15) For all $a,b\in S\cup\Gamma\cup\{0,\ldots,8\cdot\binom{p(n)}{3}\}$ with $a\neq b$, the clause $neq(a,b)$ belongs to $ext_{M,p,\sigma}$.
(16) For all $a\in S\cup\Gamma\cup\{0,\ldots,8\cdot\binom{p(n)}{3}\}$, the clause $eq(a,a)$ belongs to $ext_{M,p,\sigma}$.

The second group of facts in our extensional database $ext_{M,p,\sigma}$ is designed to help us deal with the operation of the Turing machine M when it is in the query state s_q. Our idea is to use that fact that extensions are closed under logical consequences to help us give correct answers for the oracle 3-SAT. Our idea is that we will employ a set of $p(n)$ propositional letters $A_1,\ldots,A_{p(n)}$. Recall our coding of 3-clauses $\varphi_1,\varphi_2,\ldots,\varphi_{8\cdot\binom{p(n)}{3}}$ based on the propositional letters $A_1,\ldots,A_{p(n)}$. In addition, we will employ one other propositional letter D.
Our second group of formulas in $ext_{M,p,\sigma}$ are :
(17) $content(1,i,t)\leftrightarrow\neg\varphi_i$ for all $1\leq i\leq 8\cdot\binom{p(n)}{3}$ and $0\leq t\leq p(n)$.
(18) $content(B,i)\leftrightarrow D\&\neg D$ for all $1\leq i\leq 8\cdot\binom{p(n)}{3}$ and $0\leq t\leq p(n)$.

To understand to role of the these sentences, we will briefly describe several of the predicates that will occur in ASP default theory $\langle D^{n,p}_{Trg},W^{n,p}_{Trg}\rangle$. First we will have a predicate, $tape_q(X,Y,T)$ which is to mean that at time T, symbol X is in cell Y of the query tape and a predicate $state(S,T)$ which is to mean that M is in state S at time T. We will also have two predicates $no(T)$ and $yes(T)$. The main properties that we shall prove about these predicates is that $yes(T)$ will hold if at time T if the read-write heads are in state s_q and the 3-SAT oracle gives the answer "yes" to our query and $no(T)$ will hold if at time T, if the read-write heads are in state $S=s_q$ and the 3-SAT oracle gives the answer "no" to our query.
The key to our proof that atoms $yes(T)$ and $no(T)$ correctly simulate the oracle is the fact that it will be the case that none of the atoms $content$ nor the atoms A_i occur in any conclusion of the rules of our default theory $\langle D^{n,p}_{Trg},W^{n,p}_{Trg}\rangle$. They only occur in $W^n_{Trg}\cup ext_{M,p,\sigma}$. That is, we can prove the following proposition.

Proposition 1 *Let ψ be any formula of the propositional language based on atoms $content(a,i,t)$, $a\in\{0,1\},i\leq 8\cdot\binom{n}{3}$, $0\leq t\leq p(n)$ and A_i, $1\leq i\leq p(n)$. Let T be any consistent theory generated by $W^{n,p}_{Trg}\cup ext_{M,p,\sigma}$ and any set of conclusions of rules from $D^{n,p}_{Trg}$. Then $\psi\in Th(T)$ if and only if $\psi\in Th(W^{n,p}_{Trg}\cup ext_{M,p,\sigma})$.*

Once we specify $W^n_{Trg}\cup ext_{M,p,\sigma}$, we will be able to prove via induction on the length the sequence of configurations that specify a partial computation of $M^{\text{3-SAT}}$ on input σ that the following proposition holds.

Proposition 2 *Let $\langle x_i\rangle_{i=1}^{8\cdot\binom{n}{3}}$ be a binary sequence of length $8\cdot\binom{n}{3}$. Then if M started on input σ has an accepting computation, then for all $0\leq t\leq n$, the formula $\bigvee_{i=1}^{8\cdot\binom{n}{3}}content(x_i,i,t)$ belongs to $Th(W^{n,p}_{Trg}\cup ext_{M,p,\sigma})$ if and only if the set of formulas $A=\{\neg\varphi_i:i<8\cdot\binom{n}{3},x_i=1\}$ is unsatisfiable.*

Note that Proposition 2 completely characterizes formulas of the form $\bigvee_{i=1}^{8\cdot\binom{n}{3}}content(x_i,i,t)$ that belong to $Th(W^n_{Trg}\cup ext_{M,p,\sigma})$.

Next we specify the description of $\langle D^{n,p}_{Trg}, W^{n,p}_{Trg}\rangle$ The remaining predicates of $\langle D^{n,p}_{Trg}, W^{n,p}_{Trg}\rangle$ are the following:
$tape_c(P,Q,T)$ for "the computation tape stores symbol Q at cell P at time T",
$tape_q(P,Q,T)$ for "the query tape stores symbol Q at cell P at time T",
$position_c(P,T)$ for "the c-read-write head reads the content cell P at time T",
$position_q(P,T)$ for "the q-read-write head reads the content cell P at time T",
$state(S,T)$ for "the read-write heads are in state S at time T" (notice that we have both a unary predicate $state/1$ with the content consisting of states, and $state/2$ to describe the evolution of the machine),
$yes(T)$ for the oracle gives the answer "yes" at time T,
$no(T)$ for the oracle gives the answer "yes" at time T
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T)$ for "instruction $(S1,Q1,D1,Q2,D2)$ has been selected for execution at time T",
$otherInstr(S,Q,,R,S1,Q1,D1,Q2,D2,T)$ for "instruction other than $(S1,Q1,D1,Q2,D2)$ has been selected for execution at time T",
$instr\_def(T)$ for "there is an instruction to be executed at time T",
$completion$ for "computation successfully completed", and
A, a propositional letter. [2].

The defaults in our theory $\langle D^{n,p}_{Trg}, W^{n,p}_{Trg}\rangle$ consists of rules which describe how the Turing machine M operates with an oracle for 3-SAT. That is, we have to describe how the state and the contents of the computation and query tapes evolve in the course of a computation of $M^{\text{3-SAT}}(\sigma)$. As we shall see, each individual rule is relatively simple, but there has to be a large number of rules due to the inherent complexity of describing the way a nondeterministic Turing machine evolves. The only subtle rules are the rules in Group 6 below which rely on the interaction between the predicates $content(X_i,i,T)$ described above. In the default theory $\langle D^{n,p}_{Trg}, W^{n,p}_{Trg}\rangle$, there should be no constants. That is, all constants should appear in the extensional database only. For notational convenience, we will not be strict in this respect. That is, to simplify our presentation, we will use the constants 0, f, and s_0 in $\langle D^{n,p}_{Trg}, W^{n,p}_{Trg}\rangle$. These can easily be eliminated by introducing appropriate unary predicates. Finally to simplify the clauses, we will follow here the notation used in the *smodels* syntax. That is, we shall write
$Q(\overline{X}) \leftarrow \alpha_1(\overline{X}) \wedge \cdots \wedge \alpha_n(\overline{X}) : \beta_1(\overline{X}), \ldots, \beta_m(\overline{X})$ as
$Q(\overline{X}) \leftarrow \alpha_1(\overline{X}), \ldots, \alpha_n(\overline{X}) : \beta_1(\overline{X}), \ldots, \beta_m(\overline{X})$.
Also, we will use $p(X_1;\ldots;X_k)$ as an abbreviation for $p(X_1),\ldots,p(X_k)$.

[2]The propositional letter A will be used whenever we write clauses acting as constraints. That is, the symbol A will occur in the following syntactical configuration. A will be the head of some clause, and A will also occur in the restraints of that same clause. In such situation an extension *cannot* satisfy the remaining atoms in the body of that clause.

Group 1. Defaults describing how the position of the read-write head evolves.

(1.1) (Initial position of the c-read-write head)
$position_c(P,T) \leftarrow time(T)$, $c\text{-}cell(P)$, $T=0$,
$ci\_position(P)$:

(1.2) (Initial position of the q-read-write head)
$position_q(P,T) \leftarrow time(T)$, $q\text{-}cell(P)$, $T=0$,
$qi\_position(P)$:

We have 6 rules that describe how read write heads move depending of the values of D_1 and D_2. For example, we would have the following two clauses when D_i equals l.

(1.3) $position_c(P1,T1) \leftarrow time(T;T1)$, $c\text{-}cell(P;P1)$,
$state(S;S1)$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$,
$T1 = T+1$, $P1+1=P$, $position_c(P,T)$,
$state(S,T)$, $tape_c(P,Q,T)$,
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T)$, $D1=l$, $P\neq 0$:

(1.4) $position_q(P1,T1) \leftarrow time(T;T1)$, $q\text{-}cell(P;P1)$,
$state(S;S1)$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$,
$T1 = T+1$, $P1+1=P$, $position_q(P,T)$,
$state(S,T)$, $tape_q(P,R,T)$,
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T)$, $D2=l$, $P\neq 0$:

Then we would include four more such clauses (1.5)-(1.8) to cover the cases when D_i equals r or λ.

Group 2. Defaults describing how the contents of the tape change as instructions get executed.

(2.1) $tape_c(P,Q,T) \leftarrow time(T)$, $c\text{-}cell(P)$, $symb(Q)$,
$T=0$, $data_c(P,Q)$:

(2.2) $tape_q(P,Q,T) \leftarrow time(T)$, $q\text{-}cell(P)$, $symb(Q)$,
$T=0$, $data_q(P,Q)$:

(2.3) $tape_c(P,Q1,T1) \leftarrow time(T;T1)$, $c\text{-}cell(P)$,
$state(S;S1)$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$,
$T1=T+1$, $position_c(P,T)$, $state(S,T)$, $tape_c(P,Q,T)$,
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T)$:

(2.4) $tape_q(P,Q2,T1) \leftarrow time(T;T1)$, $q\text{-}cell(P)$,
$state(S;S1)$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$,
$T1=T+1$, $position_q(P,T)$, $state(S,T)$, $tape_q(P,R,T)$,
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T)$:

(2.5) $tape_c(P,Q,T1) \leftarrow time(T;T1)$, $c\text{-}cell(P;P1)$, $symb(Q)$,
$T1=T+1$, $tape_c(P,Q,T)$, $position_c(P1,T)$, $P\neq P1$:

(2.6) $tape_q(P,Q,T1) \leftarrow time(T;T1)$, $q\text{-}cell(P;P1)$, $symb(Q)$,
$T1=T+1$, $tape_q(P,Q,T)$, $position_q(P1,T)$, $P\neq P1$:

Group 3. Defaults describing how the state of the read-write head evolves over time.

(3.1) $state(S,T) \leftarrow time(T)$, $state(S)$, $T=0$, $S=s_0$:

(3.2) $state(S1,T1) \leftarrow time(T;T1)$, $c\text{-}cell(P1)$, $q\text{-}cell(P2)$,
$state(S;S1)$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$,
$T1=T+1$, $position_c(P1,T)$, $position_q(P2,T)$,
$state(S,T)$, $tape_c(P1,Q,T)$, $tape_q(P2,R,T)$,
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T)$:

Group 4. Defaults describing the unique instruction to be executed at time T.

(4.1) Selecting instruction at step 0.
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T) \leftarrow state(S;S1)$,
$symb(Q;R;Q1;Q2)$, $dir(D1;D2)$, $time(T)$, $T=0$,
$c\text{-}cell(P1)$, $q\text{-}cell(P2)$, $S=s_0$, $tape_c(P1,Q,T)$
$ci\_position(P1)$, $tape_q(P2,R,T)$, $qi\_position(P2)$,
$delta(S,Q,R,S1,Q1,D1,Q2,D2)$:
$otherInstr((S,Q,R,S1,Q1,D1,Q2,D2,T)$

(4.2) Defaults describing the selection the instruction to be implemented at non-query steps.
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T) \leftarrow state(S;S1)$,
$symb(Q;R;Q1;Q2)$, $dir(D1;D2)$, $time(T)$, $T \neq 0$,
$c\text{-}cell(P1)$, $q\text{-}cell(P2)$, $position_c(P1,T)$, $position_q(P2,T)$,
$S \neq s_q$, $tape_c(P1,Q,T)$, $tape_q(P2,R,T)$,
$delta(S,Q,R,S1,Q1,D1,Q2,D2)$:
$otherInstr(S,Q,R,S1,Q1,D1,Q2,D2,T)$

(4.3) Selecting instruction at query steps where the oracle answers yes.
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T) \leftarrow state(S;S1)$,
$S=s_q$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$, $time(T)$,
$T \neq 0$, $c\text{-}cell(P1)$, $q\text{-}cell(P2)$, $position_c(P1,T)$,
$position_q(P2,T)$, $tape_c(P1,Q,T)$, $tape_q(P2,R,T)$,
$yes(T)$, $delta_{yes}(S,Q,R,S1,Q1,D1,Q2,D2)$:,
$otherInstr(S,Q,R,S1,Q1,D1,Q2,D2,T)$

(4.4) Selecting instruction at query steps where the oracle answers no.
$instr(S,Q,R,S1,Q1,D1,Q2,D2,T) \leftarrow state(S;S1)$,
$S=s_q$, $dir(D1;D2)$, $symb(Q;R;Q1;Q2)$, $time(T)$,
$T \neq 0$, $c\text{-}cell(P1)$, $q\text{-}cell(P2)$, $position_c(P1,T)$,
$position_q(P2,T)$, $tape_c(P1,Q,T)$, $tape_q(P2,R,T)$,
$no(T)$, $delta_{no}(S,Q,R,S1,Q1,D1,Q2,D2)$:
$otherInstr(S,Q,R,S1,Q1,D1,Q2,D2,T)$

Group 5. Defaults that define the *otherInstr* predicate. Rules (5.1)-(5.8) are designed to say that a 9-tuple $(S,Q,R,S1,Q1,D1,Q2,D2,T)$ satisfies *otherInstr* if it differs from the 9-tuple $(S',Q',R',S2,Q3,D3,,Q4,D4,T)$ that satisfies $instr$. Thus a typical rule would be

(5.1) $otherInstr(S,Q,R,S1,Q1,D1,Q2,D2,T) \leftarrow$
$state(S;S';S1;S2)$, $symb(Q;Q';R;R';Q1;Q2,Q3,Q4)$,
$time(T)$, $dir(D1;D2,D3,D4)$,
$instr(S',Q',R',S2,Q3,D3,,Q4,D4,T)$, $S \neq S'$:

Rules (5.2)-(5.8) are identical to rule (5.1) except that they end in $Q \neq Q'$:, $R \neq R'$:, $S1 \neq S2$:, $Q1 \neq Q3$:, $D1 \neq D3$:, $Q2 \neq Q4$:, $D2 \neq D4$: instead of $S \neq S'$:.

Our next two clauses are designed to ensure that exactly one instruction is selected for execution at any given time T.

(5.9) $instr\_def(T) \leftarrow state(S;S1)$, $symb(Q;Q1)$, $dir(D)$,
$time(T)$, $instr(S,Q,S1,Q1,D,T)$:

(5.10) $A \leftarrow time(T), \neg instr\_def(T) : \neg A$

Group 6. Defaults that ensure that the predicates $yes(T)$ and $no(T)$ behave properly when M is in the query state s_q at time T.

(6.1) $no(T) \leftarrow symbol(X1;\ldots:X_{8\cdot\binom{p(n)}{3}})$,
$q\text{-}cell(1;\ldots;8\cdot\binom{p(n)}{3}))$, $time(T)$, $state(s_q,T)$,
$(\bigvee_{i=1}^{8\cdot\binom{p(n)}{3}}(content(X_i,i,T))$ & $(X_i=1 \vee X_i=B)$,
$(\bigwedge_{i=1}^{8\cdot\binom{p(n)}{3}}(tape_q(X_i,i,T)$ & $(X_i=1 \vee X_i=B))$:

(6.2) $yes(T) \leftarrow symbol(X1;\ldots:X_{8\cdot\binom{p(n)}{3}})$,
$q\text{-}cell(1;\ldots;8\cdot\binom{p(n)}{3}))$, $time(T)$, $state(s_q,T)$,
$(\bigwedge_{i=1}^{8\cdot\binom{p(n)}{3}}(tape_q(X_i,i,T)$ & $(X_i=1 \vee X_i=B)) : no(T)$

The idea of these clauses is as follows. For any given time t with $0 \leq t \leq p(n)$, cell i with $0 \leq i \leq 8\cdot\binom{p(n)}{3}$ and symbol $s_{i,t} \in \{1,B\}$, the only way that we can derive $tape_q(s_{i,t},i,T)$ is if there is partial computation of $M^{\text{3-SAT}}(\sigma)$ such that $s_{i,t}$ is in cell i at of the query tape at time t. This means that to derive $no(t)$, we must be able to derive $\Theta_t = (\bigvee_{i=1}^{8\cdot\binom{p(n)}{3}}(content(s_{i,t},i,t))$. But by Proposition 1 and the clauses in our extensional database described in (17) and (18) , Θ_t can be derived from $\langle D_{Trg}, W_{Trg} \cup edb_{M,p,\sigma}\rangle$ only if $\bigvee_{s_{i,t}=1}\neg\phi_i$ is a tautology where the disjunction runs for all cells i on the query tape. But $\bigvee_{s_{i,t}=1}\neg\phi_i$ is equivalent to $\neg(\bigwedge_{s_{i,t}=1}\phi_i)$ which represents the negation of the query given to the 3-SAT oracle at time t. Since $\neg(\bigwedge_{s_{i,t}=1}\phi_i)$ is a tautology, it must be that $\bigwedge_{s_{i,t}=1}\phi_i$ is not satisfiable. Thus clause 6.1 can hold if and only if our 3-SAT oracle gives the answer no at time t. Clause 6.2 then says that if we do not get the answer no from the 3-SAT oracle at time t, then we must get the answer yes from the 3-SAT oracle at time t.

Group 7. Defaults that ensure that extensions only correspond to accepting computations.

(7.1) $completion \leftarrow symb(Q), instr(f,Q,f,Q,\lambda,p(n))$:.

(7.2) $A \leftarrow: \neg completion, \neg A$

Proposition 3 *There is a polynomial q so that for every machine M, polynomial p, and an input σ, the size of the extensional database $edb_{M,p,\sigma}$ is less than or equal to $q(|M|,|\sigma|,p(|\sigma|))$.*

We can prove that for any nondeterministic oracle Turing Machine M with oracle 3-SAT, runtime polynomial $p(x)$, and input σ of length n, the generating sets of extensions of $\langle D_{Trg}, W_{Trg} \cup edb_{M,p,\sigma}\rangle$ encode the sequences of tapes of length $p(n)$ which occur in the steps of an accepting computation of $M^{\text{3-SAT}}$ starting on σ and that any such sequence of steps can be used to produce an extension of $\langle D_{Trg}, W_{Trg} \cup edb_{M,p,\sigma}\rangle$.
The key idea is to consider valid runs of the oracle machine $M^{\text{3-SAT}}$ started on input σ. A *configuration* relative to state S is a quintuple $\langle i,U,V,u,v\rangle$ where

1. i is an instruction $\langle S,Q,R,S_1,Q_1,D_1,Q_2,D_2\rangle$,
2. U is a state of the computation tape,
3. V is the state of the query tape,
4. u is an integer $\leq p(n)$ such that $U(u) = Q \in \Sigma \cup \{B\}$, and
5. v is an integer $\leq 8\cdot\binom{p(n)}{3}$ such that $V(v) = R \in \Sigma \cup \{B\}$

Informally, u is the index of the cell on which the read-write head on the computation tape is pointing at the time the configuration is observed and $Q = U(u)$ is the content of that cell. Similarly, v is the index of the cell on which the read-write head on the query tape is pointing at the time the configuration is observed and $R = V(v)$ is the content of that cell. Conditions (4) and (5) are coherence conditions which say that the instruction i is applicable in the configuration. A valid run $\mathcal{C} = \langle C_0, \ldots, C_{p(n)} \rangle$ of the machine M, where for m, $0 \leq m \leq p(n)$,

$$C_m = \langle i_m, U_m, V_m, u_m, v_m \rangle$$

such that each transition C_i to C_{i+1} is allowed by the transitions of $M^{\text{3-SAT}}$. We define the set of atoms $N_{\mathcal{C}}$ which consists of the union of sets of atoms $N_1 \cup \ldots \cup N_{14}$ where:

$N_1 = edb_{M,p,\sigma}$

$N_2 = \{state(S,m) : i_m = \langle S, Q, R, S_1, Q_1, D_1, Q_2, D_2 \rangle$ & $0 \leq m \leq p(m)\}$

$N_3 = \{position_c(u_m, m) : 0 \leq m \leq p(n)\}$

$N_4 = \{position_q(v_m, m) : 0 \leq m \leq p(n)\}$

$N_5 = \{tape_c(r, U_m(r), m) : 0 \leq m \leq p(n),$ $0 \leq r \leq p(n) - 1\}$

$N_6 = \{tape_q(r, V_m(r), m) : 0 \leq m \leq p(n),$ $0 \leq r \leq 8 \cdot \binom{p(n)}{3} - 1\}$

$N_7 = \{yes(m) : i_m = \langle s_q, Q, R, S_1, Q_1, D_1, Q_2, D_2 \rangle$ & the set if formulas $\{\neg\varphi_i : V_m(i) = 1\}$ is satisfiable$\}$

$N_8 = \{no(m) : i_m = \langle s_q, Q, R, S_1, Q_1, D_1, Q_2, D_2 \rangle$ & the set if formulas $\{\neg\varphi_i : V_m(i) = 1\}$ is unsatisfiable$\}$

$N_9 = \{instr(S, Q, R, S_1, Q_1, D_1, Q_2, D_2, m) :$ $i_m = \langle S, Q, R, S_1, Q_1, D_1, Q_2, D_2 \rangle, 0 \leq m \leq p(n)\}$

$N_{10} = \{otherInstr(S', Q', R', S'_1, Q'_1, D'_1, Q'_2, D'_2, m) :$ $S' \neq s_q$ & $(S'_1, Q'_1, D'_1, Q'_2, D'_2) \in \delta(S', Q', R'),$ $i_m \neq \langle S', Q', R', S'_1, Q'_1, D'_1, Q'_2, D'_2 \rangle,$ $0 \leq m \leq p(n)\}$

$N_{11} = \{otherInstr(S', Q', R', S'_1, Q'_1, D'_1, Q'_2, D'_2, m) :$ $S' = s_q$ & $yes \times (S'_1, Q'_1, D'_1, Q'_2, D'_2) \in \delta(S', Q', R'),$ $i_m \neq \langle S', Q', R', S'_1, Q'_1, D'_1, Q'_2, D'_2 \rangle, 0 \leq m \leq p(n)\}$

$N_{12} = \{otherInstr(S', Q', R', S'_1, Q'_1, D'_1, Q'_2, D'_2, m) :$ $S' \neq s_q$ & $no \times (S'_1, Q'_1, D'_1, Q'_2, D'_2) \in \delta(S', Q', R'),$ $i_m \neq \langle S', Q', R', S'_1, Q'_1, D'_1, Q'_2, D'_2 \rangle, 0 \leq m \leq p(n)\}$

$N_{13} = \{instr\_def(m) : 0 \leq m \leq p(n)\}$

$N_{14} = \{completion\}$

We can then show by induction that that if $\mathcal{C}$ is valid run of $M^{\text{3-SAT}}$, then $Th(N_{\mathcal{C}})$ is an extension of $\langle D^{p,n}_{Trg}, W^{n,p}_{Trg} \cup edb_{M,p,\sigma} \rangle$ and for any extension E of $\langle D^{p,n}_{Trg}, W^{n,p}_{Trg} \cup edb_{M,p,\sigma} \rangle$, there is a valid run $\mathcal{C}$ of $M^{\text{3-SAT}}$ such that $Th(N_{\mathcal{C}}) = E$. Thus we have the following.

Theorem 2 *The mapping of Turing machines to DL Answer Set programs defined by $\langle M, \sigma, p \rangle \mapsto \langle D^{n,p}_{Trg}, W^{n,p}_{Trg} \cup edb_{M,p,\sigma} \rangle$ has the property that there is a 1-1 polynomial time correspondence between the set of generating sets $N_{\mathcal{C}}$ of extensions of $\langle M, \sigma, p \rangle \mapsto \langle D^{n,p}_{Trg}, W^{n,p}_{Trg} \cup edb_{M,p,\sigma} \rangle$ and the set of valid runs $\mathcal{C}$ of M with oracle 3-SAT of the length $p(n)$, starting on the state σ of the tape, and ending in the state f.*

Since the problem of finding accepting computations for oracle Turing machines with a 3-SAT oracle captures all Σ^P_2 search problems, it follows that computing generating sets of extensions of DL Answer Set programs captures all Σ^P_2-search problems. Thus we have the following.

Theorem 3 *The class of Σ^P_2 search problems is precisely captured by the problem of computing generating sets for extensions of DL Answer Set programs.*

IV. Conclusions

In this paper, we gave a formulation of an Answer Set Programming language based on Default Logic where the basic defaults are quantifier free and the underlying language has no function symbols. We showed that our Default Logic ASP programs capture precisely the Σ^p_2 search problems. That is, one can reduce any Σ^p_2 search problem to the problem of finding generating sets for extensions of a Default Logic ASP program and vice versa, the problem of finding a generating set for an extension of a Default Logic ASP program is a Σ^p_2 search problem. We proved our result by showing that one could uniformly code the accepting computations nondeterministic oracle Turing machines with an oracle for 3-SAT as generating sets of appropriate Default Logic ASP programs. This proof provides a precise way of explaining why questions about extensions of Default Logic theories naturally lie at the second level of polynomial time hierarchy since it shows that there is a natural way in which Default Logic theories can query an NP-complete oracle. We note that there is an alternative way to derive the same result by reducing the problem of finding generating sets for extensions of Default Logic ASP programs to the problem of answering queries of Default Logic query language of Cardoli, Eiter, and Gottlob (Cadoli et. al. 1994; Cadoli et. al. 1997).

References

N. Bidoit and Ch. Froidevaux. General Logic Databases and Programs: Default Semantics and Stratification. *Information and Computation* 19:15–54, 1991.

M. Cadoli, T. Eiter and G. Gottlob. Default Logic as a Query Language. *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning* (KR-94), pages 99–108, 1994.

M. Cadoli, T. Eiter and G. Gottlob. Default Logic as a Query Language. *IEEE Transactions on Knowledge and Data Engineering* 9:448-463, 1997.

P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński. Programming with default logic. *Artificial Intelligence Journal* 112:105–146, 1999.

D. East and M. Truszczyński. Propositional satisfiability in answer-set programming, Proceedings of Joint German/Austrian Conference on Artificial Intelligence, KI'2001, Lecture Notes in Artificial Intelligence, Springer Verlag, 2001.

T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for non-monotonic reasoning. In *Proceedings of the 4th International Conference on Logic Pro-*

gramming and Nonmonotonic Reasoning, Springer LN in Computer Science 1265, pages 363–374, 1997.

T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons, and Benchmarks. In *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 406–417, 1998.

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

M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080, 1988.

M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385, 1991.

G. Gottlob. Complexity Results for Nonmonotonic Logics. *Journal of Logic and Computation* 2:397–425, 1992.

V. Lifschitz. Answer set planning. In *Logic programming and nonmonotonic reasoning*, volume 1730 of *Lecture Notes in Computer Science*, pages 373–374. Springer-Verlag, 1999.

W. Łukaszewicz. Considerations on default logic. In R. Reiter, (ed.), *Proceedings of the International Workshop on Non-Monotonic Logic*, pages 165–193, 1984.

On the Foundations of Answer Set Programming, *Proceedings of AAAI Symposium on Answer Set Programming*, Stanford, CA. March 26-28, 2001.

V.W. Marek and M. Truszczyński. Relating Autoepistemic and Default Logics. *Proceedings of the First International Conference on Knowledge Representation and Reasoning*, pages 276–288, 1989.

V.W. Marek and M. Truszczyński. Stable Semantics for Logic Programs and Default Theories. In: *Proceedings of North American Conference on Logic Programming*, pages 243–257, 1989.

V.W. Marek and M. Truszczyński. *Nonmonotonic Logic: Context-Dependent Reasoning*. Springer Verlag, 1993.

V.W. Marek and M. Truszczyński. Stable Models and an Alternative Logic Programming Paradigm. *The Logic Programming Paradigm*, pages 375–398. Series Artificial Intelligence, Springer-Verlag, 1999.

A. Mikitiuk and M. Truszczynski. Rational Default Logic and Disjunctive Logic Programming. In: A. Nerode and L. Pereira, (eds.), *Logic Programming and Nonmonotonic Reasoning*, pages 283–299, 1993.

R. Moore. Semantical Considerations on Nonmonotonic Logic. *Artificial Intelligence Journal*, 25:75–94, 1985.

I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79, 1998.

I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of JICSLP-96*. MIT Press, 1996.

R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence Journal*, 138:181–234, 2002.

J. Stillman. The Complexity of Propositional Default Logic. *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI-92*, pages 794-799, 1992.

T. Syrjänen. Manual of Lparse version 1.0, `http://saturn.tcs.hut.fi/Software/smodels`, 2001

J. Ullman. *Principles of Database and Knowledge-Base Systems*, Computer Science Press, 1988.