Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

DEFAULT LOGIC, PROPOSITIONAL LOGIC AND CONSTRAINTS

Rachel Ben-Eliyahu
Rina Dechter

July 1991
CSD-910032

[illegible]edings of AAAI-91,
Anaheim, CA., July 1991.

TECHNICAL REPORT
R-162
April 1991

Default Logic, Propositional Logic and Constraints *

Rachel Ben-Eliyahu
< rachel@cs.ucla.edu >
Cognitive Systems Laboratory
Computer Science Department
University of California
Los-Angeles, California 90024

Rina Dechter
< dechter@ics.uci.edu >
Information & Computer Science
University of California
Irvine, California, 92717

Abstract

We present a mapping from a class of default theories to sentences in propositional logic, such that each model of the latter corresponds to an extension of the former. Using this mapping we show that many properties of default theories can be determined by solving propositional satisfiability. In particular, we show how CSP techniques can be used to identify, analyze and solve tractable subsets of Reiter's default logic.

1 Introduction

Since the introduction of Reiter's default logic [Reiter, 1980], many researchers have elaborated its semantics ([Etherington, 1987], [Konolige, 1988]) and have developed inference algorithms for default theories ([Etherington, 1987],[Kautz and Selman, 1989], [Stillman, 1990]). It was clear from the beginning that most of these computations are formidable (not even semi-decidable), and so research has focused on restricted classes of the language, searching for tractable subclasses of default theories. Unfortunately, many simplified sublanguages still remained intractable ([Kautz and Selman, 1989], [Stillman, 1990]).

Since Reiter's logic is an important formalism for nonmonotonic reasoning, it is worth exploring new dimensions along which tractable classes can be identified. The approach we propose here examines the structural features of the knowledge base, and leads to a topological characterization of nonmonotonic theories.

One language that has received a thorough topological analysis is *constraint networks*. This propositional language, based on multi-valued variables and relational constraints is also intractable, but many of its tractable subclasses have been identified by topological analysis. A constraint network is a graph (or hypergraph) in which nodes represent variables and arcs represent pairs (or sets) of variables that are included in a common constraint. The topology of such a network uncovers opportunities for problem decomposition techniques and provides estimates of the problem complexity prior to actual processing. Graphical analysis has led to the development of effective solution strategies and has identified parameters such as *width* and *cycle-cutset* that govern problem difficulty ([Freuder, 1982], [Mackworth and Freuder, 1984], [Dechter, 1990], [Dechter and Pearl, 1989]).

Our approach is to identify tractable classes of default theories by mapping them into tractable classes of constraint networks. Specifically, we reformulate a default theory within the constraint network language and use the latter to induce the appropriate solution strategies.

Rather than attempting a direct translation to constraint network, this paper describes an intermediate translation of default theories into propositional logic. Since propositional logic can be translated into constraint networks this yields a mapping from default theories to constraint networks. The intermediate translation into propositional logic may point to additional tractable classes and can shed new light on the semantics of default theories.

In the first part of this paper we show that any disjunction-free propositional default theory with semi-normal rules can be translated in polynomial time to a propositional theory such that all the interesting properties of the default theory can be computed by solving the satisfiability of the latter. In the second part we show how constraint networks can be utilized to identify tractable classes of default theories.

The paper is organized as follows. Sections 2 and 3 describe Reiter's default logic and introduce necessary notations and preliminaries. Section 4 presents our transformation and describes how tasks on a default theory are mapped into equivalent tasks in propositional logic. Section 5 discusses cyclic and ordered theories, while Section 6 presents new procedures for query processing and identifies tractable classes using constraint networks techniques. Section 7 provides concluding remarks. Due to space considerations all proofs are omitted. For more details see [Ben-Eliyahu and Dechter, 1991a].

*Supported by Air Force Office of Scientific Research, AFOSR 900136.

2 Reiter's Default Logic

Let $\mathcal{L}$ be a first order language. A *default theory* is a pair (D, W), where D is a set of defaults and W is a set of closed wffs (well formed formulas) in $\mathcal{L}$. A *default* is a rule of the form $\alpha : \beta_1, ..., \beta_n/\gamma$, where $\alpha, \beta_1, ...\beta_n$ and γ are formulas in $\mathcal{L}$. The intuition behind a default can be: If α is believed and there is no reason to believe that one of the β_i is false, then γ can be believed. A default $\alpha : \beta/\gamma$ is *normal* if $\gamma = \beta$. A default is *semi-normal* if it is in the form $\alpha : \beta \wedge \gamma/\gamma$. A default theory is *closed* if all the first order formulas in D and W are closed.

The set of defaults D induces an *extension* on W. Intuitively, an extension is a maximal set of formulas that can be deduced from W using the defaults in D. Let E^* denote the logical closure of E in $\mathcal{L}$. We use the following definition of an extension ([Reiter, 1980],theorem 2.1):

Definition 2.1 *Let $E \subseteq \mathcal{L}$ be a set of closed wffs, and let (D, W) be a closed default theory. Define*

- $E_0 = W$
- *For $i \geq 0$ $E_{i+1} = E_i^* \bigcup \{\gamma | \alpha : \beta_1, ..., \beta_n/\gamma \in D$ where $\alpha \in E_i$ and $\neg\beta_1, ...\neg\beta_n \notin E\}$*

E is an extension for (D, W) iff for some ordering $E = \bigcup_{i=0}^{\infty} E_i$. (Note the appearance of E in the formula for E_{i+1}). □

Most queries on a default theory (D, W) fall into one of the following classes:

Existence: Does (D, W) have an extension? If so, find one.

Set-Membership: Given a set of formulas S, Is S contained in some extension of (D, W)?

Set-Entailment: Given a set of formulas S, Is S contained in every extension of (D, W)?

In this paper we restrict our attention to Propositional Disjunction-free Semi-normal Default theories, denoted **PDSD** (where formulas in D and W are disjunction-free). This is the same subclass studied by Kautz and Selman [Kautz and Selman, 1989]. Clearly, when dealing with PDSDs, every extension E^* is a logical closure of a set consisting of literals only. We assume, w.l.g. that the consequent in each rule is a single literal. We can also assume, w.l.g., that W is consistent and that no default has a contradiction as a justification; when W is inconsistent, only one trivial extension exists and a default having contradictory justification can be eliminated by inspection.

3 Definitions and Preliminaries

We denote propositional symbols by upper case letters $P, Q, R...$, propositional literals (i.e. $P, \neg P$) by lower case letters $p, q, r...$ and conjunctions of literals by $\alpha, \beta....$ The operator $\sim$ over literals is defined as follows: If $p = \neg Q$, $\sim p = Q$, If $p = Q$ then $\sim p = \neg Q$.

If $\delta = \alpha : \beta/\gamma$ is a default, we define $\text{pre}(\delta) = \alpha$, $\text{just}(\delta) = \beta$ and $\text{concl}(\delta) = \gamma$.

Given a set of literals E, we say that E *satisfies the preconditions of* δ if $\text{pre}(\delta) \in E$ and for each $q \in \text{just}(\delta)$ $\sim q \notin E$ [1]. We say that E *satisfies the rule* δ if it does not satisfy the preconditions of δ or else, it satisfies both its preconditions and includes its conclusion.

A *proof of a literal* p, w.r.t. a set of literals E and a PDSD (D, W) is a sequence of rules $\delta_1, ..., \delta_n$ such that the following three conditions hold:

1. $\text{concl}(\delta_n) = p$.
2. For all $1 \leq i \leq n$ and for each $q \in \text{just}(\delta_i)$, $\neg q \notin E$
3. For all $1 \leq i \leq n$ $\text{pre}(\delta_i) \subseteq W \bigcup \{\text{concl}(\delta_1), ..., \text{concl}(\delta_{i-1})\}$.

The following lemma is instrumental throughout the paper. It can be viewed as the declarative counterpart of lemma 1 in [Kautz and Selman, 1989].

Lemma 3.1 *E^* is an extension of a PDSD (D, W) iff E^* is a logical closure of a set of literals E that satisfies:*

1. *$W \subseteq E$*
2. *E satisfies each rule in D.*
3. *For each $p \in E$, there is a proof of p in E.* □

We define the *dependency graph* $G_{(D,W)}$ of a PDSD (D, W) to be a directed graph constructed as follows: Each literal p appearing in D or in W is associated with a node, and an edge is directed from p to r iff there is a default rule where p appears in its prerequisite and r is its consequent. An *acyclic PDSD* is one whose dependency graph is acyclic, a property that can be tested linearly.

4 Expressing PDSD in Propositional Logic

The common approach for building an extension, used by [Etherington, 1987], [Kautz and Selman, 1989], and others, is to increment W using rules from D. We take a totally different approach by making a declarative account of such process: using lemma 3.1, we formulate the default theory as a set of constraints on the set of its extensions.

We first present the transformation of acyclic PDSDs and then extend it to the cyclic case.

4.1 The Acyclic Case

Extensions of acyclic PDSDs can be expressed and generated in a simpler fashion. This is demonstrated through Lemma 4.1, a relaxed version of the general Lemma 3.1. We can show that (note the change in item 3):

[1]Note that since we are dealing with PDSDs, if α is not a contradiction, the negation of one of its conjuncts is in the extension iff the negation of α is there too.

Lemma 4.1 *E^* is an extension of an* acyclic *PDSD (D, W) iff E^* is a logical closure of a set of literals E that satisfies:*

1. *$W \subseteq E$*
2. *E satisfies each rule in D.*
3. *for each $p \in E - W$ there is $\delta \in D$ such that $concl(\delta) = p$ and E satisfies the preconditions of δ.* □

Expressing the above conditions in propositional logic results in a propositional theory whose models coincide with the extensions of the acyclic default theory. Let $\mathcal{L}$ be the underlying propositional language of (D, W). For each propositional symbol in $\mathcal{L}$ we define two propositional symbols, I_P and $I_{\neg P}$, yielding a new set of symbols: $\mathcal{L}' = \{I_P, I_{\neg P} | P \in \mathcal{L}\}$. Intuitively, I_P stands for "P is in the extension" while $I_{\neg P}$ stands for "$\neg P$ is in the extension".

To simplify notations we use the notions of $in(\alpha)$ and $cons(\alpha)$ that stand for "α is in the extension", and "α is consistent with the extension", respectively. Formally, $in(\alpha)$ and $cons(\alpha)$ are defined as functions from conjuncts in $\mathcal{L}$ to conjuncts in $\mathcal{L}'$ as follows:

- if $\alpha = P$ then $in(\alpha) = I_P$, $cons(\alpha) = \neg I_{\neg P}$.
- if $\alpha = \neg P$ then $in(\alpha) = I_{\neg P}$, $cons(\alpha) = \neg I_P$.
- if $\alpha = \beta \wedge \gamma$ then $in(\alpha) = [in(\beta)] \wedge [in(\gamma)]$, $cons(\alpha) = [cons(\beta)] \wedge [cons(\gamma)]$.

The following procedure,**translate-1**, translates an acyclic PDSD (D, W) into a propositional theory $\mathcal{P}_{(D, W)}$ as follows:

translate-1$((D, W))$

1. for each $p \in W$, put I_p into $\mathcal{P}_{(D, W)}$.
2. for each $\alpha : \beta/\gamma \in D$, if $\gamma \notin W$, add $in(\alpha) \wedge cons(\beta) \rightarrow in(\gamma)$ into $\mathcal{P}_{(D, W)}$.
3. Let $S_p = \{[in(\alpha) \wedge cons(\beta)] | \exists \delta \in D$ such that $\delta = \alpha : \beta/p\}$.
   For each $p \notin W$, if $S_p \neq \emptyset$ then add to $\mathcal{P}_{(D, W)}$ the formula $I_p \rightarrow [\vee_{\alpha \in S_p} \alpha]$.
   else, (If $p \notin W$ and $S_p = \emptyset$), add to $\mathcal{P}_{(D, W)}$ the formula $\neg I_p$. □

We claim that:

Theorem 4.2 *Procedure* **translate-1** *transforms an acyclic PDSD (D, W) into a propositional theory $\mathcal{P}_{(D, W)}$ such that θ is a model for $\mathcal{P}_{(D, W)}$ iff $\{p | \theta(I_p) = \textbf{true}\}^*$ is an extension for (D, W).* □

Algorithm translate-1 is time and space linear in $|D + W|$ (assuming W is sorted).

Example 4.3 *(based on Reiter's example 2.5)*
Consider the following acyclic PDSD : $D = \{A : P/P, : A/A, \neg A/\neg A\}$, $W = \emptyset$.
$\mathcal{P}_{(D, W)} = \{$ (remains empty after step 1),
(following step 2:) $I_A \wedge \neg I_{\neg P} \rightarrow I_P$, $\neg I_{\neg A} \rightarrow I_A$, $\neg I_A \rightarrow I_{\neg A}$,
(following step 3:) $I_P \rightarrow I_A \wedge \neg I_{\neg P}$, $I_A \rightarrow \neg I_{\neg A}$, $I_{\neg A} \rightarrow \neg I_A$, $\neg I_{\neg P}\}$

$\mathcal{P}_{(D, W)}$ has only 2 models : $\{I_A = \textbf{true}, I_{\neg A} = \textbf{false}, I_{\neg P} = \textbf{false}, I_P = \textbf{true}\}$, that corresponds to the extension $\{A, P\}$, and $\{I_A = \textbf{false}, I_{\neg A} = \textbf{true}, I_{\neg P} = \textbf{false}, I_P = \textbf{false}\}$, that corresponds to the extension $\{\neg A\}$. □

4.2 The Cyclic Case

Since procedure *translate-1* assumes acyclic PDSD, it does not exclude the possibility of unfounded proofs. If applied to cyclic PDSD, the resulting transformation will possess models that correspond to illegal extensions.

Consequently, in order to adjust our translation to the cyclic case we need to strengthen the constraint in step 3 of translate-1. Namely, we must add the constraint that if a literal, not in W, belongs to the extension, then the prerequisite of at least one of its rules should be in the extension *on its own rights*, namely, not as a consequence of a circular proof. One way to avoid circular proofs is to impose indexing on literals such that for every literal in the extension there exists a proof with literals having lower indices.

To implement this idea, originally mentioned at [Dis89,], we associate an *index variable* with each literal in the transformed language, and require that p is in the extension only if it is the consequent of a rule whose prerequisite's indexes are smaller. Let $\#p$ stand for the "index associated with p", and let k be its number of values. These multi-valued variables can be expressed using k propositional literals and additional $O(k^2)$ clauses [Ben-Eliyahu and Dechter, 1991a]. For simplicity, however, we will use the multi-variable notations, viewing them as abbreviations to their propositional counterparts.

Let $\mathcal{L}''$ be the language $\mathcal{L}' \bigcup \{\#p | p \in \mathcal{L}\}$, where $\mathcal{L}'$ is the set $\{I_p, I_{\neg P} | P \in \mathcal{L}\}$ defined earlier. Procedure **translate-2** transforms any PDSD (cyclic or acyclic) over $\mathcal{L}$ to a set of propositional sentences over $\mathcal{L}''$. It is defined by modifying step 3 of translate-1 as follows:

procedure translate-2(D, W) - step 3
3. Let $C_p = \{[in(q_1 \wedge q_2 ... \wedge q_n) \wedge cons(\beta)] \wedge [\#q_1 < \#p] \wedge ... \wedge [\#q_n < \#p] | \exists \delta \in D$ such that $\delta = q_1 \wedge q_2 ... \wedge q_n : \beta/p\}$.
For each $p \notin W$, if C_p is not empty then, add to $\mathcal{P}_{(D, W)}$ the formula $I_p \rightarrow [\vee_{\alpha \in C_p} \alpha]$.
Else, (If $p \notin W$ and $C_p = \emptyset$) add $\neg I_p$ to $\mathcal{P}_{(D, W)}$. □

The complexity of this translation requires adding n index variables, n being the number of literals in $\mathcal{L}$, each having at most n values. Since expressing an *inequality* in propositional logic requires $O(n^2)$ clauses, and since there are at most n possible inequalities per default, the resulting size of this transformation is bounded by $O(|W| + |D|n^3)$ propositional sentences.

The following theorems summarize the properties of our transformation. In all of them, $\mathcal{P}_{(D,W)}$ is the set of sentences resulting from translating a given PDSD (D, W) using translate-2 (or translate-1 when the theory is acyclic).

Theorem 4.4 *Let (D, W) be a PDSD. If $\mathcal{P}_{(D,W)}$ is satisfiable and if θ is a model for $\mathcal{P}_{(D,W)}$, then $\{p|\theta(I_p) = \mathbf{true}\}^*$ is an extension for (D, W).* □

Theorem 4.5 *If E^* is an extension for (D, W) then there is a model θ for $\mathcal{P}_{(D,W)}$ such that $\theta(in(p)) =$ **true** iff $p \in E^*$.* □

Corollary 4.6 *A PDSD (D, W) has an extension iff $\mathcal{P}_{(D,W)}$ is satisfiable.* □

Corollary 4.7 *A set of literals S is contained in an extension of (D, W) iff there is a model for $\mathcal{P}_{(D,W)}$ which satisfies the set $\{I_p|p \in S\}$.* □

Corollary 4.8 *A literal p is in every extension of a PDSD (D, W) iff there is no model for $\mathcal{P}_{(D,W)}$ which satisfies $\neg I_p$.* □

The above theorems suggest that we can first translate a given PDSD (D, W) to $\mathcal{P}_{(D,W)}$ and then answer queries as follows: to test if (D, W) has an extension, we test satisfiability of $\mathcal{P}_{(D,W)}$, to see if a set S of literals is a member in some extension, we test satisfiability of $\mathcal{P}_{(D,W)} \cup \{I_p|p \in S\}$, and to see if S is included in every extension, we test if for every $p \in S$, $\mathcal{P}_{(D,W)} \cup \{\neg I_p\}$ is not satisfiable.

4.3 An Improved Translation

Procedure translate-2 can be further improved. If a prerequisite of a rule is not on a cycle with its consequent, we do not need to index them, nor to enforce the partial order among their indexes. Thus, only literals which reside on cycles in the dependency graph need indexes. Furthermore, we will never have to solve cyclicity between two literals that do not share a cycle. We show that the index variable's range can be bounded by the maximal length of an acyclic path in any ***strongly connected component*** in $G_{(D,W)}$[Ben-Eliyahu and Dechter, 1991a]. The strongly-connected components of a directed graph is a partition of its set of nodes such that for each subset C in the partition, and for each $x, y \in C$, there is a directed path from x to y and from y to x in G. The strongly connected components can be identified in linear time [Tarjan, 1972].

Procedure *translate-3* incorporates these observations by revising step 3 of translate-2. The procedure associates index variables only with literals that are part of a non-trivial cycle (i.e. cycle with at least two nodes).

```
procedure translate-3((D, W))-step 3
3.a Identify the strongly connected components of $G_{(D,W)}$.
3.b Let $S_p = \{[in(q_1 \wedge q_2 ... \wedge q_n) \wedge cons(\beta)] \wedge [\#q_1 < \#p] \wedge ... \wedge [\#q_r < \#p] \,|\exists \delta \in D$ such that $\delta = q_1 \wedge q_2 ... \wedge q_n : \beta/p$, and $q_1, ..., q_r$ $(0 \le r \le n)$ are in $p$'s component $\}$
    For each $p \notin W$ add $I_p \rightarrow [\vee_{\alpha \in S_p} \alpha]$ to $\mathcal{P}_{(D,W)}$.
    If $p \notin W$ and $S_p = \emptyset$ add $\neg I_p$ to $\mathcal{P}_{(D,W)}$. □
```

Procedure translate-3 will behave exactly as translate-1 when the input is an acyclic PDSD. The number of index variables produced by translate-3, is bounded by $\mathrm{Min}\{k*c, n\}$, where k is the size of a largest component of $G_{(D,W)}$, c is the number of non-trivial components and n the number of literals in the language. The range of the index variable is bounded by l – the length of the longest acyclic path in any component ($l \le k$). Since in each rule's prerequisite we have at most k literals that share a component with its consequence, the resulting propositional transformation is bounded by additional $O(|W| + |D|kl^2)$ sentences, giving an explicit connection between the complexity of the transformation and its cyclicity level. Theorems 4.4 through 4.8 hold for procedure translate-3 as well.

5 Acyclicity and Orderness

While we distinguish between cyclic and acyclic PDSDs, Etherington has distinguished between ordered and unordered default theories. He has defined an order induced on the set of literals by the defaults in D, and showed that if a semi-normal theory is ordered, then it has at least one extension.

To understand the relationship between these two categories we define a *generalized dependency graph* of a PDSD, to be a directed graph with blue and white arrows. Each literal is associated with a node in the graph, and for every $\delta = \alpha : \beta/p$ in D, every $q \in \alpha$, and every $r \in \beta$, there is a blue edge from q to p and a white edge from $\sim r$ to p. A PDSD is *unordered* iff its generalized dependency graph has a cycle having at least one white edge. A PDSD is *cyclic* iff its generalized dependency graph has a blue cycle (i.e., a cycle with no white edges). Therefore, a set of default rules which is ordered is not necessarily acyclic and vice versa. For instance, the set $\{P : Q/Q, Q : P/P\}$ is ordered but cyclic while the set $\{P : Q/Q, S : \neg Q \wedge P/P\}$ is acyclic but not ordered .

Clearly, the expressive power of both ordered and acyclic subsets of PDSD is restricted
[Kautz and Selman, 1989]. Cyclic theories are needed, in particular, for characterizing two properties which are highly correlated. For example, to express the belief that usually people who smoke drink and vice versa, we need the defaults Drink : Smoke /Smoke, Smoke : Drink / Drink, yielding a cyclic default theory.

The characterization of default theories presented in the following section may be viewed as a generalization of both acyclicity and orderness.

6 Topology-Based Tractability for Default Logic

What can be gained from the above transformation?

Since our translation is polynomial, if its resulting output belongs to a tractable propositional subclass, tasks of existence, set-membership and set-entailment can be performed efficiently.

One such subclass is *2-SAT*, a subclass containing disjunctions of at most two literals. The corresponding class of default theories which translates into 2-SAT was called by [Kautz and Selman, 1989] and by [Stillman, 1990] "Prerequisite free normal unary" (a PDSD with normal rules having no prerequisite). The linear satisfiability of 2-SAT induces a linear time algorithm for the corresponding class of default theories. In contrast, Kautz and Selman presented a quadratic algorithm (for deciding "membership in all extensions") applicable to a broader class of PDSDs (called "normal unary") where the prerequisite of each (normal) rule consists of a single positive literal.

Next, we view propositional satisfiability as a constraint satisfaction problem and use techniques borrowed from that field to solve satisfiability.

Constraint satisfaction techniques exploit the structure of the problem through the notion of a "constraint graph". For propositional sentences, the constraint graph (also called a "primal constraint graph") associates a node with each propositional letter and connects any two nodes whose associated letters appear in the same propositional sentence. Various graph parameters were shown as crucially related to solving the satisfiability problem. These include the *induced width*, w^*, the *size of the cycle-cutset*, the *depth of a depth-first-search spanning tree* of this graph and the *size of the non-separable components* ([Freuder, 1985]),[Dechter and Pearl, 1988], [Dechter, 1990]). It can be shown that the worse-case complexity of deciding consistency is polynomialy bounded by any one of these parameters.

Since, these parameters can be bounded easily by simple processing of the given graph, they can be used for assessing tractability ahead of time. For instance, when the constraint graph is a tree, satisfiability can be answered in linear time. In the sequel we will demonstrate the potential of this approach using one specific technique, called *Tree-Clustering* [Dechter and Pearl, 1989], customized for solving propositional satisfiability, and emphasize its effectiveness for maintaining a default data-base.

The Tree-Clustering scheme has a *tree-building* phase, and a *query processing* phase. The complexity of the former is exponentially dependent on the sparseness of the constraint graph, while the complexity of the latter is always linear in the size of the data-base generated by the tree-building preprocessing phase. Consequently, even when building the tree is computationally expensive it may be justified when many queries on the same PDSD are expected. The algorithm is summarized below (for details see [Dechter and Pearl, 1989]).

Propositional-Tree-Clustering (tree-building)

input: a set of propositional sentences S and its constraint graph.

1. Use the *triangulation algorithm* to generate a *chordal* constraint graph.

   A graph is *chordal* if every cycle of length at least four has a chord.

   The triangulation algorithm transforms any graph into a chordal graph by adding edges to it [Tarjan and Yannakakis, 1984]. It consists of two steps:

   (a) Select an ordering for the nodes, (various heuristics for good orderings are available).

   (b) Fill in edges recursively between any two nonadjacent nodes that are connected via nodes higher up in the ordering.

2. Identify all the *maximal cliques* in the graph. Let $C_1, ..., C_t$ be all such cliques indexed by the rank of their highest nodes.

3. Connect each C_i to an ancestor C_j $(j < i)$ with whom it shares the largest set of letters. The resulting graph is called a **join tree**.

4. Compute $\mathcal{M}_i$, the set of models over C_i that satisfy S_i, where S_i be the set of all sentences composed only of letters in C_i.

5. For each C_i and for *each* C_j adjacent to C_i in the join tree, delete from $\mathcal{M}_i$ every model M that has no model in $\mathcal{M}_j$ that agrees with it on the set of their common letters. This amounts to performing *arc consistency* on the join tree. □

Since the most costly operation within the *tree-building* algorithm is generating all the submodels of each clique (step 5), the time and space complexity of this preliminary phase is $O(n * 2^{|C|})$, where $|C|$ is the size of the largest clique and n is the number of letters used in S . It can be shown that $|C| = w^* + 1$, where w^* is the *width* [2] of the ordered chordal graph (also called *induced width*). As a result, for classes having a *bounded induced width*, this method is tractable.

Once the tree is built it always allows an efficient query processing. This procedure is described within the following general scenario. (n stands for the number

[2]The *width* of a node in an ordered graph is the number of edges connecting it to nodes lower in the ordering. The width of an ordering is the maximum width of nodes in that ordering, and the width of a graph is the minimal width of all its orderings

of letters in the original PDSD, m, bounds the number of submodels for each clique.) [3]

1. Translate the PDSD to propositional logic (generates $O(|W| + |D|n^3)$ sentences).
2. Build a default data-base from the propositional sentences using the *Tree-building* method (takes $O(n^2 * \exp(w^* + 1))$).
3. Answer queries on the default theory using the produced tree:
   - To answer whether there is an extension, test if there is an empty clique. If so, no extension exists (bounded by $O(n^2)$ steps).
   - To find an extension, solve the tree in a backtrack-free manner:
     In order to find a satisfying model we pick an arbitrary node C_i in the join tree, select a model M_i from $\mathcal{M}_i$, select, from each of its neighbors C_j, a model M_j that agrees with M_i on common letters, unite all these models and continue to the neighbors' neighbors, and so on. The set of all models can be generated by exhausting all combinations of submodels that agree on their common letters (finding one model is bounded by $O(n^2 * m)$ steps).
   - To answer whether there is an extension that satisfy a set of literals A, check if there is a model satisfying $\{I_p | p \in A\}$ (This takes $O(n^2 * m * logm)$ steps).
   - To answer whether a literal p is included in all the extensions, check whether there is a solution that satisfies $\neg I_p$, (bounded by $O(n^2 m)$ steps).

Following is an example demonstrating our approach.

Example 6.1 *Consider the following PDSD :*

$$D = \{ \frac{Dumbo : Elephant \wedge Fly}{Elephant} \quad \frac{Elephant \wedge \neg Fly : \neg Dumbo}{\neg Dumbo}$$
$$\frac{Elephant : \neg Fly}{\neg Fly} \quad \frac{Dumbo : Fly}{Fly}$$
$$\frac{Elephant : \neg Circus}{\neg Circus} \quad \frac{Dumbo : Elephant \wedge Circus}{Circus} \}$$
$$W = \{Dumbo, Elephant\}$$

The propositional letter "Dumbo" represents here a special kind of elephants that can fly. These defaults state that normally, Dumbos, assuming they fly, are elephants, if an elephant does not fly we do not believe that it is a Dumbo. Elephants usually do not fly, while Dumbos usually fly. Most elephants are not living in a circus while Dumbos usually live in a circus.

This is an acyclic default theory, thus algorithm *translate-1* produces the following set of sentences (each proposition is abbreviated by its initial letter):

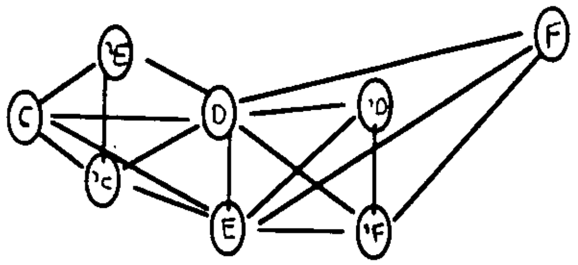

Figure 1: Constraints graph for example 6.1

Sentences generated in step 1 of translate-1: I_D, I_E.
step 2 :

$$I_E \wedge I_{\neg F} \wedge \neg I_D \rightarrow I_{\neg D},\ I_E \wedge \neg I_F \rightarrow I_{\neg F},$$
$$I_D \wedge \neg I_{\neg F} \rightarrow I_F,\ I_E \wedge \neg I_C \rightarrow I_{\neg C},$$
$$I_D \wedge \neg I_{\neg E} \wedge \neg I_{\neg C} \rightarrow I_C.$$

step 3 :

$$I_{\neg D} \rightarrow I_E \wedge I_{\neg F} \wedge \neg I_D,\ I_{\neg F} \rightarrow I_E \wedge \neg I_F,$$
$$I_F \rightarrow I_D \wedge \neg I_{\neg F},\ I_{\neg C} \rightarrow I_E \wedge \neg I_C,$$
$$I_C \rightarrow I_D \wedge \neg I_{\neg E} \wedge \neg I_{\neg C},\ \neg I_{\neg E}$$

The primal graph of this set is shown in figure 1. It is already chordal and the ordering $I_E, I_{\neg F}, I_D, I_{\neg D}, I_{\neg C}, I_C, I_F, I_{\neg E}$ suggests that for this particular problem, $w^* \leq 3$. Thus, using the *tree-Clustering* method we can answer queries about extension, set-membership and set-entailment in polynomial time (bounded by exp(4)). Note that this PDSD is unordered and not unary, therefore, the complexity of answering queries for such PDSD is NP-hard [Kautz and Selman, 1989].

We conclude this section with a characterization of the tractability of PDSD theories as a function of the topology of their *interaction graph*. The interaction graph is an undirected graph, where each literal in W or D is associated with a node and, for every $\delta = \alpha : \beta / p$ in D, every $q \in \alpha$ and every $\sim r$ such that $r \in \beta$, there are arcs connecting all of them into one clique with p.

The first theorem considers the *induced width* of the interaction graph:

Theorem 6.2 *For a PDSD (D, W) whose interaction graph has an induced width w^*, existence, membership and entailment can be decided in $O(n * 2^{w^*+1})$ steps when the theory is acyclic and $O(n^{w^*+2})$ steps when the theory is cyclic.* □

The second theorem relates the complexity to the size of the cycle cutset. A cycle cutset of a graph is a set of nodes that, once removed, would render the constraint graph cycle-free. For more details about this method, see [Dechter, 1990].

Theorem 6.3 *For a PDSD (D, W) whose interaction graph has a cycle cutset of cardinality c, existence, membership and entailment can be decided in $O(n * 2^c)$*

[3] Note that the number of letters in the propositional sentences is $O(n^2)$ if the PDSD is cyclic, and $O(n)$ if it is acyclic, and that m is bounded by the total number of extensions.

steps when the theory is acyclic and $O(n^{c+1})$ steps when the theory is cyclic. □

7 Summary and Conclusions

This paper presents a transformation of a disjunction-free semi-normal default theory into a propositional theory such that the set of models of the latter coincides with the set of extensions of the former. Questions of existence, membership and entailment posed on the default theory are thus transformed into equivalent problems of satisfiability and consistency of constraint networks. These mappings bring problems in nonmonotonic reasoning into the familiar arenas of propositional satisfiability and constraint satisfaction problems.

Using our transformation, we showed that default theories whose interaction graph has a bounded $w*$ are tractable, and can be solved in time and space bounded by $O(n^{w*+2})$ steps. This permits us to predict worse-case performance prior to processing, since w^* can be bounded in time quadratic in the number of literals. Moreover, the tree-clustering procedure, associated with the w^* analysis, provides an effective preprocessing strategy for maintaining the knowledge; Once applied, all incoming queries can be answered swiftly and changes to the knowledge can often be incorporated in linear time. Similar results were established relative to a second parameter - the cardinality of the cycle cutset.

In the full paper we elaborate on these and on additional tractable classes identified by CSP techniques like cycle-cutset, non-separable component and backjumping. In [Ben-Eliyahu and Dechter, 1991b] we have extended the results presented in this paper to "network default theories", defined by Etherington, in which W contains clauses of size less or equal to two. We believe that our transformation can be carried over to general disjunctive semi-normal default theories as well.

Acknowledgment

We thank Judea Pearl for helpful comments on earlier versions of this paper and Caroline Ehrlich for proof reading it.

References

[Ben-Eliyahu and Dechter, 1991a] Rachel Ben-Eliyahu and Rina Dechter. Expressing default theories in constarint language, 1991. in preparation.

[Ben-Eliyahu and Dechter, 1991b] Rachel Ben-Eliyahu and Rina Dechter. Inference in inheritance networks using propositional logic and constraints networks techniques. Technical Report R-163, Cognitive systems lab, UCLA, 1991.

[Dechter and Pearl, 1988] Rina Dechter and Judea Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[Dechter and Pearl, 1989] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[Dis89,] Paul Morris suggested it in a discussion following the constranits processing workshop in AAAI-89.

[Etherington, 1987] David W. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence*, 31:41–85, 1987.

[Freuder, 1982] E.C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982.

[Freuder, 1985] E.C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, 1985.

[Kautz and Selman, 1989] Henry A. Kautz and Bart Selman. Hard problems for simple default logics. In *KR-89*, pages 189–197, Toronto,Ontario,Canada, 1989.

[Konolige, 1988] Kurt Konolige. On the relation between default and autoepistemic logic. *Artificial Intelligence*, 35:343–382, 1988.

[Mackworth and Freuder, 1984] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–74, 1984.

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

[Stillman, 1990] Jonathan Stillman. It's not my default : The complexity of membership problems in restricted propositional default logics. In *AAAI-90*, pages 571–578, Boston,MA, 1990.

[Tarjan and Yannakakis, 1984] Robert E. Tarjan and M Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM journal of Computing*, 13(3):566–579, 1984.

[Tarjan, 1972] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal of Computing*, 1(2), June 1972.