J. Komorowski Z.W. Raś (Eds.)

Methodologies for Intelligent Systems

7th International Symposium, ISMIS '93
Trondheim, Norway, June 15-18, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editor

Jörg Siekmann
University of Saarland
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, W-6600 Saarbrücken 11, FRG

Volume Editors

Jan Komorowski
Knowledge Systems Group
Faculty of Computer Science and Electrical Engineering
The Norwegian Institute of Technology, The University of Trondheim
N-7034 Trondheim, Norway

Zbigniew W. Raś
Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA

CR Subject Classification (1991): I.2

ISBN 3-540-56804-2 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56804-2 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

Typesetting: Camera ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

This volume contains papers which were selected for presentation at the Seventh International Symposium on Methodologies for Intelligent Systems - ISMIS'93, held in Trondheim, Norway, June 15-18, 1993. The symposium was hosted by the Norwegian Institute of Technology and sponsored by The University of Trondheim, NFR/NTNF – The Norwegian Research Council, UNC-Charlotte, Office of Naval Research, Oak Ridge National Laboratory and ESPRIT BRA Compulog Network of Excellence.

ISMIS is a conference series that was started in 1986 in Knoxville, Tennessee. It has since then been held in Charlotte, North Carolina, once in Knoxville, and once in Torino, Italy.

The Organizing Committee has decided to select the following major areas for ISMIS'93:

- Approximate Reasoning
- Constraint Programming
- Expert Systems
- Intelligent Databases
- Knowledge Representation
- Learning and Adaptive Systems
- Manufacturing
- Methodologies

The contributed papers were selected from more than 120 full draft papers by the following Program Committee: Jens Balchen (NTH, Norway), Alan W. Biermann (Duke, USA), Alan Bundy (Edinburgh, Scotland), Jacques Calmet (Karlsruhe, Germany), Jaime Carbonell (Carnegie-Mellon, USA), David Hislop (US Army Research Office), Eero Hyvonen (VTT, Finland), Marek Karpinski (Bonn, Germany), Yves Kodratoff (Paris VI, France), Jan Komorowski (NTH, Norway), Kurt Konolige (SRI International, USA), Catherine Lassez (Yorktown Heights, USA), Lennart Ljung (Linköping, Sweden), Ramon Lopez de Mantaras (CSIC, Spain), Alberto Martelli (Torino, Italy), Ryszard Michalski (George Mason, USA), Jack Minker, (Maryland, USA), Rohit Parikh (CUNY, USA), Judea Pearl (UCLA, USA), Don Perlis (Maryland, USA), Francois G. Pinn (ORNL, USA), Henri Prade (Toulouse, France), Zbigniew W. Raś (UNC, USA), Barry Richards (Imperial College, UK), Colette Rolland (Paris I, France), Lorenza Saitta (Trento, Italy), Erik Sandewall (Linköping, Sweden), Richmond Thomason (Pittsburgh, USA), Enn Tyugu (KTH, Sweden), Ralph Wachter (ONR, USA), S.K. Michael Wong (Regina, Canada), Erling Woods (SINTEF, Norway), Maria Zemankova (NSF, USA) and Jan Zytkow (Wichita State, USA). Additionally, we acknowledge the help in reviewing the papers from: M. Beckerman, Sanjiv Bhatia, Jianhua Chen, Stephen Chenoweth, Bill Cheu, Bipin Desai, Keith Downing, Doug Fisher, Melvin Fitting, Theresa Gaasterland, Atillio Giordana, Charles Glover, Diana Gordon, Jerzy Grzymala-Busse, Cezary Janikow, Kien-Chung Kuo, Rei-Chi Lee, Charles Ling, Anthony Maida, Stan Matwin,

Neil Murray, David Mutchler, Jan Plaza, Helena Rasiowa, Steven Salzberg, P.F. Spelt, David Reed, Michael Sobolewski, Stan Szpakowicz, Zbigniew Stachniak, K. Thirunarayan, Marianne Winslett, Agata Wrzos-Kamińska, Jacek Wrzos-Kamiński, Jing Xiao, Wlodek Zadrozny and Wojtek Ziarko.

The Symposium was organized by the Knowledge Systems Group of the Department of Computer Systems and Telematics, The Norwegian Institute of Technology. The Congress Department of the Institute provided the secretariat of the Symposium. The Organizing Committee consisted of Jan Komorowski, Zbigniew W. Raś and Jacek Wrzos-Kamiński.

We wish to express our thanks to François Bry, Lennart Ljung, Michael Lowry, Jack Minker, Luc De Raedt and Erik Sandewall who presented the invited addresses at the symposium. We would also like to express our appreciation to the sponsors of the symposium and to all who submitted papers for presentation and publication in the proceedings. Special thanks are due to Alfred Hofmann of Springer Verlag for his help and support.

Finally, we would like to thank Jacek Wrzos-Kamiński whose contribution to organizing this symposium was essential to its becoming a success.

March 1993 J. Komorowski, Z.W. Raś

Table of Contents

Logic for Artificial Intelligence II

Intelligent Databases

Invited Talk III

Logic for Artificial Intelligence III

Approximate Reasoning

Invited Talk IV

Constraint Programming

Learning and Adaptive Systems I

Invited Talk V

Methodologies

Knowledge Representation

On Extended Disjunctive Logic Programs

Jack Minker[1,2] and Carolina Ruiz[1]

[1] Department of Computer Science.
[2] Institute for Advanced Computer Studies.
University of Maryland. College Park, MD 20742 U.S.A.
{minker , cruizc}@cs.umd.edu

Abstract. This paper studies, in a comprehensive manner, different aspects of extended disjunctive logic programs, that is, programs whose clauses are of the form $l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$, where $l_1, ..., l_n$ are literals (i.e. atoms and classically negated atoms), and *not* is the negation-by-default operator. The explicit use of classical negation suggests the introduction of a new truth value, namely, *logical falsehood* (in contrast to *falsehood-by-default*) in the semantics. General techniques are described for extending the model, fixpoint, and proof theories of an arbitrary semantics of normal disjunctive logic programs to cover the class of extended programs. Illustrations of these techniques are given for stable models, disjunctive well-founded and stationary semantics. Also, the declarative complexity of the extended programs as well as the algorithmic complexity of the proof procedures are discussed.

1 Introduction

Logic programming, as an approach to the use of logic in knowledge representation and reasoning, has gone through different stages. First, logic programs containing only Horn clauses were considered. A *Horn clause* is a disjunction of literals in which at most one literal is positive and can be written either as: "$a \leftarrow b_1, \ldots, b_m$" or as "$\leftarrow b_1, \ldots, b_m$" where $a, b_1, \ldots, b_m$ are atoms and $m \geq 0$. The semantics of these programs is well understood (see [31, 15]) and is captured by the unique minimal Herbrand model of the program.

It is clear that since only positive atoms occur in (the head of) Horn clauses, no negative information can be inferred from these programs unless some strategy or rule for deriving negative information is adopted. Two rules for negation were initially proposed for Horn programs: The *Closed World Assumption* (CWA) [28] which states that an atom can be assumed to be *false* if it cannot be proven to be *true*; and the *Clark completion theory* [7] which assumes that the definition of each atom in a program is complete in the sense that it specifies all the circumstances under which the atom is *true* and only such circumstances, so the atom can be inferred *false* otherwise.

Having a rule for negation, it is plausible to extend Horn clauses to make use of negative information. This is the purpose of the so-called *negation-by-default* operator *not*, which may appear in the bodies of clauses. These clauses are called *normal clauses* and are of the form: "$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$" where

$a, b_1, \ldots, b_m, c_1, \ldots, c_n$ are atoms and $m, n \geq 0$. This kind of negation is limited, however, in the sense that *not* p does not refer to the presence of knowledge asserting the falsehood of the atom p but only to the lack of evidence about its truth. Indeed, some authors have translated *not* p as "p is not believed" [14], "p is not known" [10], and "there is no evidence that p is *true*" [9], in addition to the common translation "p is not provable from the program in question".

In contrast to the Horn case, there is no agreement on a unique semantics for normal programs since there can be as many different semantics as there are ways to interpret the meaning of *not*. Among the proposed semantics are the perfect model semantics [24], the stable model semantics [11], and the well-founded semantics (WFS) [32].

Another generalization of Horn clauses that allows disjunctions of atoms in the heads of clauses has been studied extensively (see [16]). These clauses are called *disjunctive clauses* and are of the following form: "$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m$" where $a_1, \ldots, a_k, b_1, \ldots, b_m$ are atoms and $k, m \geq 0$. The meaning of such a program is captured by its set of minimal Herbrand models. Several rules for negation have been introduced for disjunctive logic programs: *the Generalized Closed World Assumption* (GCWA) [20] which assumes that an atom is *false* when it does not belong to any of the minimal Herbrand models of the program, *the Extended Generalized Closed World Assumption* (EGCWA) [33] which applies exactly the same criterion of the GCWA but to conjunctions of atoms instead of only atoms (see Sect. 4) and *the Weak Generalized Closed World Assumption* (WGCWA) [27] (or equivalently, the *Disjunctive Database Rule (DDR)* [29]) which states that an atom can be assumed to be *false* when it does not appear in any disjunction derivable from the program.

Negative information can be introduced in disjunctive clauses in the same fashion as in Horn clauses. The resulting clauses are called *normal disjunctive clauses* and are of the form: "$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$" where $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are atoms and $k, m, n \geq 0$. There are also various different semantics proposed for normal disjunctive logic programs (henceforth, denoted by *ndlps*), among others, the stable disjunctive model semantics [23], the disjunctive well-founded semantics (DWFS) [2], the generalized disjunctive well-founded semantics (GWFS) [3, 4], WF^3 [5], and the stationary semantics [26].

It is worth noting that normal clauses are particular cases of disjunctive normal clauses. Therefore any semantics defined for the class of normal disjunctive logic programs is also a semantics for the class of normal logic programs.

An alternative to overcome some of the difficulties of dealing with negative information is to make explicit use of classical negation in addition to negation-by-default. In this way, the expressive power of logic programs is increased since the user is now allowed to state not only when an atom is *true* but also when it is *false* (without any ambiguity or default interpretation). Clauses obtained by explicitly using the classical negation operator ($\neg$) are called *extended disjunctive clauses* and are of the following form: "$l_1 \vee \ldots \vee l_k \leftarrow l_{k+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$," where $l_1, \ldots, l_n$ are literals (i.e. atoms and classically negated atoms), $0 \leq k \leq$

$m \leq n$. Hence, extended disjunctive clauses contain two forms of negation: classical and default.

Previous contributions in this area include the following: Pearce and Wagner [22] added explicit negative information to Prolog programs. They showed that there is no need to alter the computational structure of such programs to include classical negation since there is a way to transform extended programs to positive ones which preserves the meaning of the programs. Gelfond and Lifschitz [12] extended their stable model semantics to cover classical negation. Przymusinski [25] generalized this extended version of the stable model semantics to include disjunctive programs. Alferes and Pereira [1] provided a framework to compare the behavior of the different semantics in the presence of two kinds of negation.

The purpose of this paper is to study, in a comprehensive manner, different aspects of extended disjunctive logic programs (*edlps* for short). We describe general techniques to deal with this extended class of programs and also survey some of the results in the field.

Alternative semantics for edlps can be obtained by extending the semantics known for the class of normal disjunctive logic programs. Since there are now two different notions of falsehood in extended programs we distinguish between them by saying that, with respect to some semantics, a formula φ is *false-by-default* in an edlp P if *not* (φ) is provable from P, i.e. φ is assumed to be *false* by the particular rule for negation used by the semantics; and is *logically false* (or simply *false*) if $\neg\varphi$ is provable from P, or in other words, if $\neg\varphi$ is a logical consequence of P. We extend each semantics to include a new truth value: *logical falsehood*.

With the introduction of negated atoms in the heads of the clauses, it is possible to specify inconsistent theories, that is, to describe situations in which some atom p and its complement $\neg p$ are *true* simultaneously. Therefore, we must develop techniques to recognize when a program is inconsistent with respect to a given semantics and to deal with such an inconsistent program.

Since the techniques to be explained are general enough to be applied to any semantics of ndlps we will describe them in terms of a generic such semantics which we call *SEM*. In addition, we will illustrate the application of these techniques to the stable model semantics (covering in this way the perfect model semantics), DWFS (which covers the WFS and the minimal models semantics for disjunctive logic programs), and the stationary semantics.

The paper is organized as follows: Section 2 introduces the notation and definitions needed in the following sections. Section 3 describes a standard procedure to extend the model theoretical characterization of an arbitrary semantics of ndlps to the whole class of edlps. It includes also an illustration of this technique for the the case of the stable model semantics. Section 4 constructs a fixpoint operator to compute the extended version of a semantics *SEM* in terms of a fixpoint operator which computes the restriction of this semantics to ndlps. Illustrations are given for the DWFS and the stationary semantics. Section 5 describes a procedure to answer queries with respect to edlps and an arbitrary semantics *SEM*. This procedure uses as a subroutine, a procedure to answer

queries with respect to the restriction of *SEM* to ndlps. Section 6 studies the complexities of some fundamental problems related to edlps.

2 Syntax and Definitions

In this section we formalize the definition of extended disjunctive logic programs and introduce some of the notation needed in the following sections.

An *extended disjunctive logic program, edlp* , is a (possibly infinite) set of clauses of the form: $l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$, where $l_1, ..., l_n$ are literals (i.e. atoms and classically negated atoms), $0 \leq k \leq m \leq n$ and *not* is the negation-by-default operator.

Example 1. The following is an extended disjunctive logic program:

$$\begin{aligned} P = \{\ & a \vee e\ ; \\ & c \leftarrow a\ ,\ not\ b\ ; \\ & \neg b \leftarrow \neg e\ ; \\ & b \leftarrow e\ ,\ not\ c\ ; \\ & \neg a \leftarrow not\ a \qquad \}\ . \end{aligned}$$

We assume the convention that any occurrence of $\neg\neg p$ is simplified to p.

Since a non-ground clause is equivalent to the set of all its ground instances, we consider here only ground programs (i.e. propositional programs). This is done only to simplify the notation without any loss of generality.

Given a program P, L_P denotes the *set of predicate symbols* that occur in P; $\mathcal{L}$ denotes the *set of all ground literals* that can be constructed with predicates in L_P; and $\mathcal{U}$ will denote the *Herbrand universe* associated with L_P.

In the context of ndlps, DHB_P (resp. CHB_P) denotes the *disjunctive Herbrand base* (resp. *conjunctive Herbrand base*) of P, that is, the set of equivalence classes of disjunctions (resp. conjunctions) of atoms appearing in P modulo logical equivalence. This notion is generalized to edlps by $\mathcal{DL}_P$ (resp. $\mathcal{CL}_P$), the set of equivalence classes of disjunctions (resp. conjunctions) of literals in $\mathcal{L}$ modulo logical equivalence.[3]

As noted before, extended programs enable us not only to state when a predicate p holds but also when $\neg p$ holds. In this sense, one can regard p and $\neg p$ as different predicates which happen to be complementary (i.e. they cannot both be *true* or both be *false* at once). Using this idea, Pearce and Wagner in [22] and Gelfond and Lifschitz in [13] showed how to transform extended normal clauses into normal clauses. This is done by representing every negative literal $\neg p$ in a program by a new predicate, say p', with the restriction that p and p' cannot hold simultaneously. This restriction may be viewed as an integrity constraint.

Formally, we define the *prime transformation* l' of a literal l to be:

$$l' = \begin{cases} p, & \text{if } l = p \text{ for some predicate } p \\ p', & \text{if } l = \neg p \text{ for some predicate } p \end{cases}$$

[3] For simplicity, we will write d as an abbreviation for the equivalence class $[d]$.

Notice that if we apply this prime transformation to every literal occurring in an edlp P, we obtain a *normal* disjunctive logic program P'. The union of P' with the following set of integrity constraints captures the same meaning of P:

$$IC_{P'} = \{\Leftarrow p, p' : p \in L_P\}$$

These integrity constraints state that p and p' are in fact complementary predicates. We use here the symbol $\Leftarrow$ instead of $\leftarrow$ to emphasize that these integrity constraints are not clauses of the program, i.e. $IC_{P'}$ is not contained in P'.

In the same spirit, $\mathcal{L}'$ denotes the set of prime literals $\{l' : l \in \mathcal{L}\}$ and will be taken as the set of predicate symbols appearing in P', i.e. $L_{P'} =_{def} \mathcal{L}'$.

Sometimes we need to recover program P from P'. In order to do so, we define the *neg transformation* on predicates by:

$$l^\neg = \begin{cases} p, & \text{if } l = p \text{ for some predicate } p \\ \neg p, & \text{if } l = p' \text{ for some predicate } p \end{cases}$$

which is extended to programs in the usual way.

It is clear that for any edlp P, $(P')^\neg = P$ and for any ndlp Q, $(Q^\neg)' = Q$. Also, it is worth noting that the prime transformation is not strictly needed. Instead of performing the prime transformation, we can treat $\neg a$ as if it is an atom independent of a. However, we will use this transformation in order to make explicit when an edlp P is thought of as a normal disjunctive logic program.

3 Model Theory Semantics

In this section we describe a standard procedure to extend an arbitrary model theory semantics of normal disjunctive logic programs to the whole class of extended disjunctive logic programs. Since the procedure is general enough to be applied to any semantics defined on the class of ndlps we describe it in terms of a generic such semantics which we call SEM. In the following subsection we illustrate the use of the technique when SEM is the stable model semantics.

We denote by *interpretation* any subset of the set of literals $\mathcal{L}$, and we call an interpretation *consistent* only if it does not contain any pair of complementary literals, say p and $\neg p$. The prime and neg transformations of interpretations are defined as expected: if $M \subseteq \mathcal{L}$ then $M' = \{l' : l \in M\}$ and if $N \subseteq \mathcal{L}'$ then $N^\neg = \{l^\neg : l \in N\}$.

Interpretations which agree with a given program (in the sense of the following definition) are called *models* of the program.

Definition 1. Let P be an edlp and let $M \subseteq \mathcal{L}$. Then M is a *model* of P iff M is consistent and for each program clause $l_1 \vee ... \vee l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ in P, if $l_{k+1}, ..., l_m \in M$ and $l_{m+1}, ..., l_n \notin M$ then $\exists i$, $1 \leq i \leq k$, such that $l_i \in M$.

The following Lemma establishes some relationships between the models of an edlp P and the models of P'.