Outline

Classical Negation: Syntax

Generalisation

Extend the language of Logic Programs to allow classical negation $\neg$ (for atoms only!), besides default negation *not* (or $\sim$).

Definition (Language)

Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg A \mid A \in \mathcal{A}\}$.

- We assume $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$.
- The atoms A and $\neg A$ are complementary.
    - $\neg A$ is the classical negation of A, and vice versa.

Classical Negation: Semantics

Definition (Consistency)

A set X of atoms (over $\mathcal{A} \cup \overline{\mathcal{A}}$) is consistent if $X \cap \{A \mid \neg A \in X\} = \emptyset$, and inconsistent, otherwise.

Definition (Answer Set)

A set X of atoms is an answer set of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ if X is an answer set of $\Pi \cup \{B \leftarrow A, \neg A \mid A \in \mathcal{A}, B \in (\mathcal{A} \cup \overline{\mathcal{A}})\}$

Proposition

For a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:

1. *All answer sets of Π are consistent or*
2. *$X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only answer set of Π.*

Classical Negation: Examples

Example

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

Classical Negation: Translation

Definition ((Possibly inconsistent) answer sets)

For determining the (possibly inconsistent) answer sets of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ in the standard way, translate Π into Π' as follows:

$$\Pi' \quad = \quad \Pi \cup \{B \leftarrow A, \neg A \ \ \neg B \leftarrow A, \neg A \mid A \in \mathcal{A}, B \in \mathcal{A}, A \neq B\}$$

Definition (Consistent answer sets)

In order to determine the answer sets of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ in the standard way, translate Π (or $\mathcal{F}$) into Π'' (or $\mathcal{F}''$) as follows:

$$\Pi'' \quad = \quad \Pi \cup \{\leftarrow A, \neg A \mid A \in \mathcal{A}\}$$

Example

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \textit{not } r\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p \,;\, q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{q\}$
- $\Pi = \{p \,;\, \textit{not } p \leftarrow \top,\ \neg p \,;\, \textit{not } q \leftarrow \top,\ q \,;\, \textit{not } q \leftarrow \top\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$
  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Language Extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, e.g., classical negation.
- This translation might also be used for implementing the language extension. When is this feasible?

Outline

Choice Rules

Idea

Choices over subsets.

Syntax

$$\{A_1, \ldots, A_m\} \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o,$$

Informal meaning

If the body is satisfied in an answer set, then any subset of $\{A_1, \ldots, A_m\}$ can be included in the answer set.

Example

The program $\Pi = \{\ \{a\} \leftarrow b,\ b \leftarrow\}$ has two answer sets: $\{b\}$ and $\{a, b\}$.

Implementation

lparse/gringo/i-dlv + smodels/cmodels/nomore/clasp/wasp

Choice Rules: Embedding in normal logic programs

Definition (Embedding of Choice Rules in normal logic programs)

A choice rule of form

$$\{A_1, \ldots, A_m\} \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o$$

can be translated into $2m + 1$ rules

$$\begin{array}{ccc} & A \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o. & \\ A_1 \leftarrow A, \mathit{not}\ \overline{A_1}. & \ldots & A_m \leftarrow A, \mathit{not}\ \overline{A_m}. \\ \overline{A_1} \leftarrow \mathit{not}\ A_1. & \ldots & \overline{A_m} \leftarrow \mathit{not}\ A_m \end{array}$$

by introducing new atoms $A, \overline{A_1}, \ldots, \overline{A_m}$.

Outline

Cardinality constraints

Syntax

A (positive) cardinality constraint is of the form $l \; \{A_1, \ldots, A_m\} \; u$

Informal meaning

A cardinality constraint is satisfied in an answer set X, if the number of atoms from $\{A_1, \ldots, A_m\}$ satisfied in X is between l and u (inclusive).
More formally, if $l \leq |\{A_1, \ldots, A_m\} \cap X| \leq u$.

Conditions

$l \; \{A_1 : B_1, \ldots, A_m : B_m\} \; u$ where $B_1, \ldots, B_m$ are used for restricting instantiations of variables occurring in $A_1, \ldots, A_m$.

Example

2 {hd(a),...,hd(m)} 4

Implementation

lparse/gringo/i-dlv + smodels/cmodels/nomore/clasp/wasp

n-colorability revisited (with $n = 3$)

Example (n-colorability (with $n = 3$))

C(I)	vertex(1) $\leftarrow$ edge(1,2) $\leftarrow$ vertex(2) $\leftarrow$ edge(2,3) $\leftarrow$ vertex(3) $\leftarrow$ edge(3,1) $\leftarrow$
C(P)	color(r) $\leftarrow$ color(b) $\leftarrow$ color(g) $\leftarrow$ 1 {colored(V,C) : color(C)} 1 $\leftarrow$ vertex(V) $\leftarrow$ edge(V,U),color(C), colored(V,C),colored(U,C)
Answer set	{ colored(1,r), colored(2,b), colored(3,g), ... }

Outline

Cardinality rules

Idea

Control cardinality of subsets.

Syntax

$$A_0 \leftarrow l \; \{A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\}$$

Informal meaning

If at least l elements of the "body" are true in an answer set, then add A_0 to the answer set. l is a lower bound on the "body"

Example

The program $\Pi = \{\ a \leftarrow 1\{b, c\},\ b \leftarrow\}$ has one answer set: $\{a, b\}$.

Implementation

lparse/gringo/i-dlv + smodels/cmodels/nomore/clasp/wasp

Cardinality Rules: Embedding in normal logic programs

Definition (Embedding of Cardinality Rules in normal logic programs)

Replace each cardinality rule

$$A_0 \leftarrow l\ \{A_1, \ldots, A_m\} \qquad \text{by} \qquad A_0 \leftarrow cc(1, l)$$

where atom $cc(i, j)$ represents the fact that at least j of the atoms in $\{A_i, \ldots, A_m\}$, that is, of the atoms that have an index equal or greater than i, are in a particular answer set.
The definition of $cc(i, j)$ is given by the rules

$$\begin{array}{rcl} cc(i, j+1) & \leftarrow & cc(i+1, j), A_i \\ cc(i, j) & \leftarrow & cc(i+1, j) \\ cc(m+1, 0) & \leftarrow & \end{array}$$

What about space complexity? The problem is that if the set $\{A_1, ..., A_m\}$ is big, then for this quadratic translation the resulting set of rules is rather large, requiring $O(ml)$ new atoms to be introduced. Moreover, the size of the translation grows towards $O(m^2)$ with the value of l.

Normal Rules: Embedding in Cardinality Rules

Definition (Normal Rules: Embedding in Cardinality Rules)

A normal rule

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n,$$

can be represented by the cardinality rule

$$A_0 \leftarrow n + m\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\}.$$

Cardinality Rules with upper bounds

Definition (Embedding of Cardinality Rules with upper bounds in normal logic programs)

A rule of the form

$$A_0 \leftarrow l\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\}\ u$$

stands for

$$\begin{array}{rcl} A_0 & \leftarrow & B, not\ C \\ B & \leftarrow & l\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\} \\ C & \leftarrow & u+1\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\} \end{array}$$

Cardinality Constraints as heads

Definition (Embedding of Cardinality Constraints as heads in normal logic programs)

A rule of the form

$$l\ \{A_1, \ldots, A_m\}\ u \leftarrow A_{m+1}, \ldots, A_n, not\ A_{n+1}, \ldots, not\ A_o,$$

stands for

$$\begin{array}{rcl}
B & \leftarrow & A_{m+1}, \ldots, A_n, not\ A_{n+1}, \ldots, not\ A_o \\
\{A_1, \ldots, A_m\} & \leftarrow & B \\
C & \leftarrow & l\ \{A_1, \ldots, A_m\}\ u \\
 & \leftarrow & B, not\ C
\end{array}$$

Full-fledged Cardinality Rules

Definition (Embedding of Cardinality Rules in normal logic programs)

A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \ldots, l_n\ S_n\ u_n$$

stands for $0 \leq i \leq n$

$$\begin{array}{rcl}
B_i & \leftarrow & l_i\ S_i \\
C_i & \leftarrow & u_i + 1\ S_i \\
A & \leftarrow & B_1, \ldots, B_n, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \\
 & \leftarrow & A, \mathit{not}\ B_0 \\
 & \leftarrow & A, C_0 \\
S_0 \cap \mathcal{A} & \leftarrow & A
\end{array}$$

where $\mathcal{A}$ is the underlying alphabet.

Outline

Weight constraints

Syntax

$$l\ [A_1 = w_1, \ldots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n]\ u$$

Informal meaning

A weight constraint is satisfied in an answer set X, if

$$l \leq \left(\sum_{1 \leq i \leq m, A_i \in X} w_i + \sum_{m < i \leq n, A_i \notin X} w_i\right) \leq u\ .$$

Generalization of cardinality constraints.

Example

10 [course(1)=3,course(2)=6,. . . ,course(10)=9] 20

Implementation

lparse/gringo/i-dlv + smodels/cmodels/nomore/clasp/wasp

Optimization statements

Idea

Compute optimal answer sets by minimizing or maximizing a weighted sum of given atoms, respectively.

Syntax

$$minimize\ [A_1 = w_1, \ldots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n]$$
$$maximize\ [A_1 = w_1, \ldots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n]$$

Several optimization statements are interpreted lexicographically.

Example

- maximize [course(1)=3,course(2)=6,. . . ,course(10)=9]
- minimize [road(X,Y) : length(X,Y,L) = L]

Implementation

lparse/gringo/i-dlv + smodels/cmodels/nomore/clasp/wasp

Weak integrity constraints

Syntax

$$:\sim A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\ [w : l]$$

Informal meaning

1. minimize the sum of weights of violated constraints in the highest level;
2. minimize the sum of weights of violated constraints in the next lower level;
3. etc

Implementation

dlv (i-dlv + wasp)

Conditional literals in gringo

- We often want to encode the contents of a (multi-)set rather than enumerating each of the elements.
- To support this, `lparse` and `gringo` allow for conditional literals.

Syntax

$$A_0 : A_1 : \ldots : A_m : \mathit{not}\ A_{m+1} : \ldots : \mathit{not}\ A_n$$

Informal meaning

List all ground instances of A_0 such that corresponding instances of $A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$ are true.

Example

`gringo` instantiates the program:

```
p(1). p(2). p(3).  q(2).
{r(X) : p(X) : not q(X)}.
```

to:

```
p(1). p(2). p(3).  q(2).
{r(1), r(3)}.
```

Domain predicates in `gringo`

- The predicates of literals on the right-hand side of a colon (:) must be defined from facts without any negative recursion.
- Such domain predicates are fully evaluated by `gringo`.

Example

```
p(1). p(2).
q(X) :- p(X), not p(X+1).
q(X) :- p(X), q(X+1).
r(X) :- p(X), not r(X+1).
```

- `p/1` and `q/1` are domain predicates because none of them negatively depends on itself.
- `r/1` is not a domain predicate because it is defined in terms of `not r(X+1)`.

See `gringo` documentations for further details.

Outline

Aggregates: Motivation

- Aggregates provide a general way to obtain a single value from a collection of input values given as a set, a bag, or a list.
- Popular aggregate (functions):
  - Average
  - Count
  - Maximum
  - Minimum
  - Sum
- Cardinality and Weight constraints rely on Count and Sum aggregates.

Aggregates: Syntax

Definition (Aggregate)

An aggregate has the form:

$$F \ \langle A_1 = w_1, \ldots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n \rangle \prec k$$

where

- F stands for a function mapping multi-sets of $\mathbb{Z}$ to $\mathbb{Z} \cup \{+\infty, -\infty\}$,
- $\prec$ stands for a relation between $\mathbb{Z} \cup \{+\infty, -\infty\}$ and $\mathbb{Z}$,
- k is an integer,
- A_i are atoms, and
- w_i are integers

for $1 \leq i \leq n$.

Example

$sum \ \langle course(1) = 3, course(2) = 6, \ldots, course(10) = 9 \rangle \leq 60$

Definition (Semantics of Aggregates)

- A (positive) aggregate $F\,\langle A_1 = w_1, \ldots, A_n = w_n \rangle \prec k$ can be represented by the formula:

$$\bigwedge_{I \subseteq \{1,\ldots,n\}, F\langle w_i | i \in I \rangle \not\prec k} \left(\bigwedge_{i \in I} A_i \rightarrow \bigvee_{i \in \bar{I}} A_i \right)$$

  where $\bar{I} = \{1, \ldots, n\} \setminus I$ and $\not\prec$ is the complement of $\prec$.
- Then, $F\,\langle A_1 = w_1, \ldots, A_n = w_n \rangle \prec k$ is true in X iff the above formula is true in X.

Aggregates: An example

Example

- Consider $sum\langle p = 1, q = 1\rangle \neq 1$
  - i.e, $A_1 = p$, $A_2 = q$ and $w_1 = 1$, $w_2 = 1$

I	$\langle w_i \mid i \in I\rangle$	$sum\langle w_i \mid i \in I\rangle$	$sum\langle w_i \mid i \in I\rangle = 1$
$\emptyset$	$\langle\rangle$	0	*false*
$\{1\}$	$\langle 1\rangle$	1	*true*
$\{2\}$	$\langle 1\rangle$	1	*true*
$\{1,2\}$	$\langle 1,1\rangle$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\langle p = 1, q = 1\rangle = 1$.

Recall

$$\bigwedge_{I\subseteq\{1,\ldots,n\}, F\langle w_i | i\in I\rangle \not\prec k} \left(\bigwedge_{i\in I} A_i \rightarrow \bigvee_{i\in \bar{I}} A_i \right)$$

Aggregates: Monotonicity

Monotone aggregates

- For instance,
  - $body^+(r)$
  - $sum\langle p = 1, q = 1\rangle > 1$ amounts to $q \land p$
- We get a simpler characterization: $\bigwedge_{I \subseteq \{1,\dots,n\}, F\langle w_i | i \in I\rangle \not\prec k} \bigvee_{i \in \bar{I}} A_i$

Anti-monotone aggregates

- For instance,
  - $body^-(r)$
  - $sum\langle p = 1, q = 1\rangle < 1$ amounts to $\neg p \land \neg q$
- We get a simpler characterization: $\bigwedge_{I \subseteq \{1,\dots,n\}, F\langle w_i | i \in I\rangle \not\prec k} \neg \bigwedge_{i \in I} A_i$

Non-monotone aggregates

- For instance, $sum\langle p = 1, q = 1\rangle \neq 1$ is non-monotone.

Bibliography I

C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

S. Brass and J. Dix.
Semantics of (disjunctive) logic programs based on partial evaluation.
Journal of Logic Programming, 40(1):1–46, 1999.

P. Cabalar and P. Ferraris.
Propositional theories are strongly equivalent to logic programs.
Theory and Practice of Logic Programming, 7(6):745–759, 2007.

K. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

Bibliography II

E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
ACM Computing Surveys, 33(3):374–425, 2001.

T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

Bibliography III

T. Eiter, M. Fink, H. Tompits, and S. Woltran.
Simplifying logic programs under uniform and strong equivalence.
In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Artificial Intelligence*, pages 87–99. Springer-Verlag, 2004.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Recursive aggregates in disjunctive logic programs: Semantics and complexity.
In José Júlio Alferes and João Alexandre Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 2004.

P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, pages 615–664. College Publications, 2005.

Bibliography IV

Paolo Ferraris and Vladimir Lifschitz.
Mathematical foundations of answer set programming.
In Sergei N. Artëmov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! (1)*, pages 615–664. College Publications, 2005.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz.
A new perspective on stable models.
In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 372–379, 2007.

Paolo Ferraris.
Answer sets for propositional theories.
In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 2005.

Bibliography V

H Gaifman and E. Shapiro.
Fully abstract compositional semantics for logic programs.
In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 134–142, 1989.

M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.
Available at `http://potassco.sourceforge.net`.

Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf.
The well-founded semantics for general logic programs.
J. ACM, 38(3):620–650, 1991.

M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.
In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. The MIT Press, 1988.

Bibliography VI

M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In *Proceedings of the International Conference on Logic Programming*, pages 579–597, 1990.

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

H. Kautz and B. Selman.
Planning as satisfiability.
In B. Neumann, editor, *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363. John Wiley & sons, 1992.

R. Kowalski.
Logic for data description.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 77–103. Plenum Press, 1978.

Bibliography VII

Joohyung Lee, Vladimir Lifschitz, and Ravi Palla.
A reductive semantics for counting and choice in answer set programming.
In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 472–479. AAAI Press, 2008.

Joohyung Lee.
A model-theoretic counterpart of loop formulas.
In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 503–508. Professional Book Center, 2005.

N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

Bibliography VIII

V. Lifschitz and H. Turner.
Splitting a logic program.
In *Proceedings of the Eleventh International Conference on Logic Programming*, pages 23–37. MIT Press, 1994.

V. Lifschitz, L. Tang, and H. Turner.
Nested expressions in logic programs.
Annals of Mathematics and Artificial Intelligence, 25(3-4):369–389, 1999.

V. Lifschitz, D. Pearce, and A. Valverde.
Strongly equivalent logic programs.
ACM Transactions on Computational Logic, 2(4):526–541, 2001.

Vladimir Lifschitz, David Pearce, and Agustín Valverde.
Strongly equivalent logic programs.
ACM Trans. Comput. Log., 2(4):526–541, 2001.

V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

Vladimir Lifschitz.
Twelve definitions of a stable model.
In Maria Garcia de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2008.

Fangzhen Lin and Yuting Zhao.
Assat: computing answer sets of a logic program by sat solvers.
Artif. Intell., 157(1-2):115–137, 2004.

Fangzhen Lin and Yi Zhou.
From answer set logic programming to circumscription via logic of gk.
In Manuela M. Veloso, editor, *IJCAI*, pages 441–446, 2007.

Bibliography X

J. Lloyd.
Foundations of Logic Programming.
Symbolic Computation. Springer-Verlag, 2nd edition, 1987.

J. McCarthy and P. J. Hayes.
Some philosophical problems from the standpoint of artificial intelligence.
pages 26–45, 1987.

John McCarthy.
Circumscription - a form of non-monotonic reasoning.
Artif. Intell., 13(1-2):27–39, 1980.

Drew V. McDermott and Jon Doyle.
Non-monotonic logic i.
Artif. Intell., 13(1-2):41–72, 1980.

Drew V. McDermott.
Nonmonotonic logic ii: Nonmonotonic modal theories.
J. ACM, 29(1):33–57, 1982.

Bibliography XI

- M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry.
  An A-prolog decision support system for the space shuttle.
  In I. Ramakrishnan, editor, *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 2001.

- E. Oikarinen and T. Janhunen.
  Modular equivalence for normal logic programs.
  In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, pages 412–416. IOS Press, 2006.

- M. Osorio, J. Navarro, and J. Arrazola.
  Equivalence in answer set programming.
  In A. Pettorossi, editor, *Proceedings of the Eleventh International Workshop on Logic Based Program Synthesis and Transformation (LOPSTR'01)*, volume 2372 of *Lecture Notes in Computer Science*, pages 57–75. Springer-Verlag, 2001.

David Pearce, Hans Tompits, and Stefan Woltran.
Encodings for equilibrium logic and logic programs with nested expressions.
In Pavel Brazdil and Alípio Jorge, editors, *EPIA*, volume 2258 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2001.

David Pearce.
A new logical characterisation of stable models and answer sets.
In Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski, editors, *NMELP*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 1996.

David Pearce.
Equilibrium logic.
Ann. Math. Artif. Intell., 47(1-2):3–41, 2006.

Raymond Reiter.
A logic for default reasoning.
Artif. Intell., 13(1-2):81–132, 1980.

Bibliography XIII

Domenico Saccà and Carlo Zaniolo.
Stable models and non-determinism in logic programs with negation.
In *PODS*, pages 205–217. ACM Press, 1990.

P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

T. Syrjänen.
Lparse 1.0 user's manual.
http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.

H. Turner.
Strong equivalence made easy: nested expressions and weight constraints.
Theory and Practice of Logic Programming, 3(4-5):609–622, 2003.

Bibliography XIV

Maarten H. van Emden and Robert A. Kowalski.
The semantics of predicate logic as a programming language.
J. ACM, 23(4):733–742, 1976.