[18] G. Plotkin. Call by name, call by value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[19] J. Rees and W. Clinger. *Revised$^3$ Report on the Algorithmic Language Scheme, SIGPLAN Notices, vol. 21, n. 12*, 1986.

[20] T. Sato and H. Tamaki. Existential continuations. *New Generation Computing*, (6):421–438, 1989.

[21] D. A. Schmidt. *Denotational semantics: a methodology for language development.* Allyn and Bacon Inc, 1986.

[22] D. S. Scott. Data types as lattices. *SIAM Journal of Computing*, 5:522–587, 1976.

[23] P. Tarau and M. Boyer. Elementary logic programs. In P. Deransart and J. Maluszynski, editors, *Proc. International Workshop on Programming languages implementation and Logic Programming - LNCS 456.* Springer-Verlag, 1990.

[24] K. Ueda. Making exhaustive search programs deterministic: Part II. In J.L Lassez, editor, *Proc. 4-th International Conference on Logic Programming*, Melbourne, 1987. MIT Press.

# A Temporal Logic for Program Specification

Manuel Enciso  Inmaculada P. de Guzmán  Carlos Rossi

Facultad de Informática. Universidad de Málaga
Pza. El Ejido s/n. 29013 Málaga, Spain
email: gimac@ctima.uma.es

**Abstract**

The introduction of the temporal analysis in Logic has stimulated different approaches, some of them artificially opposed, such as to consider an absolute or relative nature of time, to consider points or intervals or to consider different time flows ... In this paper we develop a temporal logic that combines these approaches to have a logic with a good computational behaviour. This logic, which we call LNint, is a modal logic that combines the treatment of points and intervals and declarations about dates and dated intervals like temporal logics with temporal arguments, or like reified logics and consequently, we obtain a mixture of the absolute and relative approaches to the treatment of time.

Our logic has a high level of expressiveness. This is a basic property in program specifications. In this work, we use the LNint logic to specify a method to control concurrent accesses over a database. In this way, we show that our logic is suitable for specifications.

## 1 Introduction

The application of temporal logic to control program execution aims to take advantage of the potential of these logics to share naturally the declarative and imperative programming paradigms within a single system. The most significant contribution to this line of work is, undoubtedly, [8] in which the Until-Since-Fixed Point Operator logic (USF) is used to establish the possibility of having both a declarative and imperative view of the temporal logic formulas.

Although until now temporal logic of points has been the most commonly used in computing, recently there exists an increasing tendency to work with interval-based temporal logics because they appear to be more adequate for reasoning about *change* ([2], [10], [14], [11], etc.), temporal reasoning in general ([1], [4]), or database management ([6]).

A point-based temporal logic has several advantages, perhaps the most important one is that it inherits the well-known computational behaviour of point algebra ([16]). However, we conclude that we did not really have to choose between one kind of temporal logic or another, because in practice, each logic form, can satisfy naturally the needs of the application we have in mind ([15]).

In this paper, we have chosen to treat points and intervals jointly to obtain a logic that adequately describes *changes, events, states* and *processes* and that can specify computational problems associated with these ontological entities.

In this work we suggest the use of the LNint [12] modal temporal logic over discrete time not as a query or data manipulation language, but more as a metalanguage to specify the control of program execution. We discuss the utility of LNint in applications to computational problems, based on the following arguments:

- LNint has natural temporal connectives that correspond to interpretations that represent interesting properties in real systems. Besides, LNint combines points and intervals, and the absolute and relative approaches, providing a high level of expressiveness.
- The LNint semantics (called "topological semantics"), based on the next and the last occurrence of a formula, allows us to define easily the meaning of temporal connectives and to simplify the proof of the metatheory.

The work content is as follows: Section 2 introduces the LNint logic. Section 3 describes the optimistic method to resolve the problem of the concurrent execution over transactions of a database. Section 4 shows the solution to this problem provided by LNint with the philosophy of the technique referred to above.

# 2 The LNint Logic

LNint is a modal temporal logic for handling points and intervals. At the syntactic level, we begin with two components, one to collect assertions about: points, points that belong to intervals and dates; and other collecting *events*. We use atomic events in a similar sense to that used by Allen, i.e., expressions about intervals which are not true at the subintervals nor, more specifically, at the points of the interval over which the expression is affirmed. These initial components, although separated at first, will be extended later to collect mixed concepts, to reach finally a perfect semantic cohabitation. This cohabitation takes shape in the following idea: we talk about points or intervals in LNint, but we are always in an (evaluation) point, the *current instant* or present. Later, we abstract this instant. In this way, we can avoid the ambiguous concept of the *current interval* that the interval modal logics impose.

LNint permits a better absolute treatment of time than other modal temporal logics, because we can deal with fixed instants and intervals, or concrete dates. This allows us to handle points and intervals, when needed, like temporal logics with temporal arguments, or like reified logics. In this way, we obtain a mixture of the absolute and relative approaches to the treatment of time.

Our reason for using LNint is to obtain the benefits of combination of several approaches to represent temporal knowledge.

Before we present the syntax and semantics of LNint, we must point out that the time flow in LNint is $(T, \leq, +)$ isomorphic to $(\mathbb{Z}, \leq, +)$.

## 2.1 Syntax of LNint

*The Alphabet:*



The alphabet is formed by the following symbols:

1. An enumerable set $\Omega_p = \{p, q, \ldots, p_1, q_1, \ldots, p_n, q_n, \ldots\}$ of point atoms that formalize statements whose executions take place in an instant.

   For example, *"the robot receives an interruption signal"* (assuming that the signal is instantaneous).

2. A set $\underline{T} = \{\underline{t} \mid t \in T\}$ of *date* atoms is used to name known instants.

   Thus, $\underline{t}$ can symbolize *instant 8*, or *year 8 b. C.*, or *April the 3rd of 1969* according to the needs of the application in which we are working.

3. An enumerable set $\Omega_{int} = \{\alpha, \beta, \ldots, \alpha_1, \beta_1, \ldots, \alpha_n, \beta_n, \ldots\}$ of atomic events (in the sense explained above).

   For example, $\alpha$ may symbolize the statements *"I have run 100 m."*, or *"the robot executed the X routine (completely)"*; these statements will be untrue at the points of the interval over which we affirm them.

4. The symbols $\top$ and $\bot$, to denote truth and falsity, respectively.

5. The punctuation symbols $\uparrow, \downarrow, \rightarrow$ , [, ] and "," .

   We will use this symbols $\uparrow, \downarrow$ and $\rightarrow$ to combine points and events.

6. The set of boolean connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

7. The symbols of point temporal binary connectives $\preccurlyeq$ and $\succcurlyeq$

   $\preccurlyeq$ and $\succcurlyeq$ are the primitive connectives of LN [5]

8. The symbols of interval temporal unary connectives $\langle ab^+ \rangle$, $\langle ab^- \rangle$, $\langle beg \rangle$, $\langle \widehat{beg} \rangle$, $\langle end \rangle$, $\langle \widehat{end} \rangle$.

These are primitive connectives taken of Halpern and Shoham logic.

*Well-formed formulas (wffs):*

We aim to create a language that allows us to talk simultaneously of points and intervals. This means that we must define several sublanguages [1]:

### I. The "events language" $\mathcal{L}_{int}$

This sublanguage helps us treat events in two different ways:

- *relatively*, by expressing the temporal relation between events. Allen [2] states that there are twelve possible temporal relations between two intervals, although Shoham [14] later shows six are sufficient because the other six can de defined from the first six; $\langle ab^+ \rangle$, $\langle ab^- \rangle$, $\langle beg \rangle$, $\langle \widehat{beg} \rangle$, $\langle end \rangle$, $\langle \widehat{end} \rangle$ the six basic connectives of our alphabet.
- *absolutely*, by linking events with intervals of known extremes (dates).

[1] In our notation, we denote the point well-formed formulas with uppercase letters $(A, B, \ldots)$ and the well-formed formulas of $\mathcal{L}_{int}$ with uppercase calligraphic letters $(\mathcal{A}, \mathcal{B}, \ldots)$.

$\mathcal{L}_{int}$ is defined inductively as follows:

- If $\alpha \in \Omega_{int}$, then $\alpha$ is a wff of $\mathcal{L}_{int}$.
- $\top$ and $\bot$ are wffs of $\mathcal{L}_{int}$.
- If $\mathcal{A}$ and $\mathcal{B}$ are wffs of $\mathcal{L}_{int}$ , then $\neg\mathcal{A}$, $\mathcal{A} \wedge \mathcal{B}$, $\mathcal{A} \vee \mathcal{B}$, $\mathcal{A} \rightarrow \mathcal{B}$, and $\mathcal{A} \leftrightarrow \mathcal{B}$ are also wffs of $\mathcal{L}_{int}$.
- If $\mathcal{A}$ is a wff of $\mathcal{L}_{int}$ , then $\langle ab^{+}\rangle\mathcal{A}$, $\langle ab^{-}\rangle\mathcal{A}$ , $\langle beg\rangle\mathcal{A}$, $\langle \widehat{beg}\rangle\mathcal{A}$, $\langle end\rangle\mathcal{A}$, and $\langle \widehat{end}\rangle\mathcal{A}$ are also wffs of $\mathcal{L}_{int}$.
- If $\underline{m}, \underline{n} \in \underline{T}$, with $m < n$, then $[\underline{m}, \underline{n}]$ is a wff of $\mathcal{L}_{int}$.

**Note:** The last item is one of the assumptions we make in this paper, because we considered only non-instantaneous intervals and events. We consider declarations about intervals of zero duration as declarations about points. On the other hand, we consider that, according to the needs of the applications on which we are working, the statement "the 1st July 1993" can be a declaration that refers to an instant and, thus, the statement "july 1993 $=_{\text{def}}$ [1st July 1993, 31st July 1993]" is a declaration that refers to interval, specifically, is an event (in the sense that Allen uses).

We consider that the temporal connectives have the same priority, and a higher priority than the binary boolean connectives.

As an example, we describe the intuitive meaning of some unary connectives of the events:

- " $\langle ab^{+}\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_2, t_3]$ abutting on the right (future) of $[t_1, t_2]$."
- " $\langle beg\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_1, t_3]$ which is a strict initial sub-interval of $[t_1, t_2]$."
- " $\langle end\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_3, t_2]$ which is a strict final sub-interval of $[t_1, t_2]$."
- " $\langle ab^{-}\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_3, t_1]$ abutting on the left (past) of $[t_1, t_2]$."
- " $\langle \widehat{beg}\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_1, t_3]$ which is a strict initial super-interval of $[t_1, t_2]$."
- " $\langle \widehat{end}\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_3, t_2]$ which is a strict final super-interval of $[t_1, t_2]$."

In $\mathcal{L}_{int}$ we introduce the connectives defined below to collect the six temporal relations remaining between intervals. The definitions are as follows:

$$\begin{array}{llll} \langle dur\rangle\mathcal{A} & =_{\text{def}} & \langle beg\rangle\langle end\rangle\mathcal{A} & \langle \widehat{dur}\rangle\mathcal{A} \\ \langle aft\rangle\mathcal{A} & =_{\text{def}} & \langle ab^{+}\rangle\langle ab^{+}\rangle\mathcal{A} & \langle bef\rangle\mathcal{A} \\ \langle over^{+}\rangle\mathcal{A} & =_{\text{def}} & \langle end\rangle\langle \widehat{beg}\rangle\mathcal{A} & \langle over^{-}\rangle\mathcal{A} \end{array} \quad \begin{array}{ll} =_{\text{def}} & \langle \widehat{beg}\rangle\langle \widehat{end}\rangle\mathcal{A} \\ =_{\text{def}} & \langle ab^{-}\rangle\langle ab^{-}\rangle\mathcal{A} \\ =_{\text{def}} & \langle beg\rangle\langle \widehat{end}\rangle\mathcal{A} \end{array}$$

These connectives are read in the usual way, for example:



- "$\langle dur\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_3, t_4]$ such that $t_1 < t_3 < t_4 < t_2$"
- "$\langle over^{+}\rangle\mathcal{A}$ is true at an interval $[t_1, t_2]$ if $\mathcal{A}$ is true at an interval $[t_3, t_4]$ such that $t_1 < t_3 < t_2 < t_4$"

**Note:** We define later the connective that collects the linking between events and dated intervals , because we have not yet the assembled elements.

## II. The "points language" $\mathcal{L}_p$.

Our intention was to formalize with this language the declarations about points, and the declarations about intervals inherited by the points belonging to intervals and the declarations about dates.

$\mathcal{L}_p$ is based on the language of the LN logic, extended to talk about dates. The inductive definition of the well-formed formulas of $\mathcal{L}_p$ is as follows:

- If $p \in \Omega_p$, then $p$ is a wff of $\mathcal{L}_p$.
- If $\underline{t} \in \underline{T}$, then $\underline{t}$ is a wff of $\mathcal{L}_p$.
- $\top$ and $\bot$ are wffs of $\mathcal{L}_p$.
- If $A$ and $B$ are wffs of $\mathcal{L}_p$, then $\neg A$ , $A \wedge B$, $A \vee B$, $A \rightarrow B$ and $A \leftrightarrow B$ also are wffs of $\mathcal{L}_p$.
- If $A$ and $B$ are wffs of $\mathcal{L}_p$, then $A \preccurlyeq B$ and $A \succcurlyeq B$ also are wffs of $\mathcal{L}_p$.

We describe the temporal connectives informally as follows:

$A \preccurlyeq B$ is read as *sometime in the future A, and the next occurrence of A will be before or simultaneous with the next occurrence of B.*

$A \succcurlyeq B$ is read as *sometime in the past A, and the last occurrence of A was after or simultaneous with the last occurrence of B.*

Just as in $\mathcal{L}_{int}$, we consider that the temporal connectives have the same priority, and a higher priority than the binary boolean connectives.

Now, we introduce the following collection of connectives defined in $\mathcal{L}_p$:

| | | | |
|---|---|---|---|
| $\oplus A$ | $=_{\text{def}}$ | $A \preccurlyeq \top$ | (read as *A will be true tomorrow*) |
| $\ominus A$ | $=_{\text{def}}$ | $A \succcurlyeq \top$ | (read as *A was true yesterday*) |
| $FA$ | $=_{\text{def}}$ | $A \preccurlyeq A$ | (read as *A will sometime be true*) |
| $PA$ | $=_{\text{def}}$ | $A \succcurlyeq A$ | (read as *A was sometime true*) |
| $GA$ | $=_{\text{def}}$ | $\neg(\neg A \preccurlyeq \neg A)$ | (read as *A will always be true*) |
| $HA$ | $=_{\text{def}}$ | $\neg(\neg A \succcurlyeq \neg A)$ | (read as *A was always true*) |

$A \prec B =_{\text{def}} A \preccurlyeq B \wedge \neg(B \preccurlyeq A)$ (read as *sometime in the future A, and the next occurrence of A will be before the next occurrence of B*)

$A \succ B =_{\text{def}} A \succcurlyeq B \wedge \neg(B \succcurlyeq A)$ (read as *sometime in the past A, and the last occurrence of A was after the last occurrence of B*)

$A \approx^{+} B =_{\text{def}} A \preccurlyeq B \wedge B \preccurlyeq A$ (read as *sometime in the future, A and B will occur and the next occurrence of A will be simultaneous with the next occurrence of B*)

$A \approx^{-} B =_{\text{def}} A \succcurlyeq B \wedge B \succcurlyeq A$ (read as *sometime in the past A and B occurred, and the last occurrence of A was simultaneous with the last occurrence of B*)

$A \; atnext \; B \;\; =_{\text{def}} \;\; B \approx^{+} (A \wedge B)$ (read as *A will be true at the next instant of time that B occurs*)

$A \; atlast \; B \;\; =_{\text{def}} \;\; B \approx^{-} (A \wedge B)$ (read as *A was true at the last instant of time that B occurred*)

In addition, to have the absolute approach available to treat time, we define:

$$A \; at \; \underline{m} =_{\text{def}} (A \wedge \underline{m}) \vee A \; atnext \; \underline{m} \vee A \; atlast \; \underline{m}$$

This connective tells us if a formula $A$ is true at a given instant; it is independent of the instant in which we are located, i. e., we deliberately ignore the current instant, and this allows us to make declarations (in first order LNint) like, for example [2], "*the president of USA in 1962 died in 1963* ", that we can represent as:

$$Pr(?, USA) \; at \; \underline{1962} \rightarrow D(?) \; at \; \underline{1963}$$

## III. The "points and events language" $\widehat{\mathcal{L}_p}$.

Now that we have defined $\mathcal{L}_{int}$ and $\mathcal{L}_p$, to avoid the intrinsic use of the interval language, we need to extend $\mathcal{L}_p$ to treat the events by using points. Of course, although it is true that events in themselves have no sense in the points of the interval over which we affirm them, it is also true that every event must have a *start instant* and an *end instant*, and that the event *takes place* at every point between them. The foregoing consideration is evident for a discrete flow of time and it can be extended rigorously for dense or continuous time flows, using the notions of *real-present*, *real-past* and *real-future* introduced in [7]. From these considerations, we can now characterize events by their start and end instants, and the points of their course. They are formalized in the extension of $\mathcal{L}_p$ which we denote as $\widehat{\mathcal{L}_p}$ which we define continuously as:

For each atomic event $\alpha \in \Omega_{int}$ we consider the following three propositions:

- $\uparrow\alpha$ is read at a given point in time as: *this is the starting instant of the event* $\alpha$.
- $\downarrow\alpha$ is read at a given point in time as: *this is the ending instant of the event* $\alpha$.
- $\overrightarrow{\alpha}$ is read at a given point in time as: *this is an instant at which the event* $\alpha$ *is happening.*

[2] Adapted example from [3].



For example, let $\alpha$ be the atomic event "*the robot executes the X routine (completely)*", then:

- $\uparrow\alpha$ is true at $t$ if *the robot starts the execution of the X routine* at the instant $t$.
- $\downarrow\alpha$ is true at $t$ if *the robot finishes the execution of the X routine* at the instant $t$.
- $\overrightarrow{\alpha}$ is true at $t$ if $t$ lies strictly inside the interval during which *the robot is executing the X routine.*

Now we define $\Omega_p^I$ as follows:

$$\Omega_p^I = \Omega_p \cup \underline{T} \cup \{\top, \bot\} \cup \{\uparrow\alpha, \downarrow\alpha, \overrightarrow{\alpha} \mid \alpha \in \Omega_{int}\}$$

We define $\widehat{\mathcal{L}_p}$ as the inductive closure of $\Omega_p^I$ under the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\preccurlyeq$ and $\succcurlyeq$. The connectives $\oplus$, $\ominus$, $F$, $P$, $G$, $H$, *atnext*, *atlast*, $\prec$, $\succ$, $\approx^{+}$, $\approx^{-}$ and *at* are defined in the same way as in $\mathcal{L}_p$.

The next step is the most important in the construction process of our definitive language, since it allows us to extend the definition ( in $\widehat{\mathcal{L}_p}$) of the start instant, the end instant and the course to the well-formed formulas of $\mathcal{L}_{int}$. This facilitates the effective manipulation of any event based on its start and end instants. Hence we obtain:

Let $\star \in \{\wedge, \vee\}$

$$\begin{aligned}
\overrightarrow{[\underline{m}, \underline{n}]} &=_{\text{def}} P\underline{m} \wedge F\underline{n} \\
\overrightarrow{(A \star B)} &=_{\text{def}} \overrightarrow{A} \star \overrightarrow{B} \\
\overrightarrow{(\neg A)} &=_{\text{def}} \neg(\uparrow A \vee \downarrow A \vee \overrightarrow{A}) \\
\overrightarrow{\langle ab^{+}\rangle A} &=_{\text{def}} F\uparrow A \\
\overrightarrow{\langle ab^{-}\rangle A} &=_{\text{def}} P\downarrow A \\
\overrightarrow{\langle beg\rangle A} &=_{\text{def}} \overrightarrow{A} \vee \downarrow A \vee P\downarrow A \\
\overrightarrow{\langle \widehat{beg}\rangle A} &=_{\text{def}} \overrightarrow{A} \wedge \oplus\overrightarrow{A} \\
\overrightarrow{\langle end\rangle A} &=_{\text{def}} \overrightarrow{A} \vee \uparrow A \vee F\uparrow A \\
\overrightarrow{\langle \widehat{end}\rangle A} &=_{\text{def}} \overrightarrow{A} \wedge \ominus\overrightarrow{A}
\end{aligned}$$

$$\begin{aligned}
\uparrow[\underline{m}, \underline{n}] &=_{\text{def}} \underline{m} \\
\uparrow(\neg A) &=_{\text{def}} \downarrow A \\
\uparrow(A \vee B) &=_{\text{def}} (\uparrow A \vee \uparrow B) \wedge \ominus(\neg\overrightarrow{A} \wedge \neg\overrightarrow{B}) \\
\uparrow(A \wedge B) &=_{\text{def}} (\uparrow A \wedge \uparrow B)
\end{aligned}$$

$$\begin{aligned}
\downarrow[\underline{m}, \underline{n}] &=_{\text{def}} \underline{n} \\
\downarrow(\neg A) &=_{\text{def}} \uparrow A \\
\downarrow(A \vee B) &=_{\text{def}} (\downarrow A \vee \downarrow B) \wedge \oplus(\neg\overrightarrow{A} \wedge \neg\overrightarrow{B}) \\
\downarrow(A \wedge B) &=_{\text{def}} \downarrow A \wedge \downarrow B
\end{aligned}$$

$$\begin{array}{llll} \uparrow\langle ab^{+}\rangle\mathcal{A} & =_{\text{def}} & F\uparrow\mathcal{A} & \downarrow\langle ab^{+}\rangle\mathcal{A} =_{\text{def}} \uparrow\mathcal{A} \\ \uparrow\langle ab^{-}\rangle\mathcal{A} & =_{\text{def}} & \downarrow\mathcal{A} & \downarrow\langle ab^{-}\rangle\mathcal{A} =_{\text{def}} P\downarrow\mathcal{A} \\ \uparrow\langle beg\rangle\mathcal{A} & =_{\text{def}} & \uparrow\mathcal{A} & \downarrow\langle beg\rangle\mathcal{A} =_{\text{def}} \downarrow\mathcal{A}\succ\uparrow\mathcal{A} \\ \uparrow\langle\widehat{beg}\rangle\mathcal{A} & =_{\text{def}} & \uparrow\mathcal{A} & \downarrow\langle\widehat{beg}\rangle\mathcal{A} =_{\text{def}} \overrightarrow{\mathcal{A}} \\ \uparrow\langle end\rangle\mathcal{A} & =_{\text{def}} & \uparrow\mathcal{A}\prec\downarrow\mathcal{A} & \downarrow\langle end\rangle\mathcal{A} =_{\text{def}} \downarrow\mathcal{A} \\ \uparrow\langle\widehat{end}\rangle\mathcal{A} & =_{\text{def}} & \overrightarrow{\mathcal{A}} & \downarrow\langle\widehat{end}\rangle\mathcal{A} =_{\text{def}} \downarrow\mathcal{A} \end{array}$$

Now, we have all the elements we need for to establish the final language of LNint, which we shall define as follows:

### IV. The language $\mathcal{L}$ of LNint.

To establish the final language, $\mathcal{L}$, of LNint, we define a translation function $\mathcal{T}r$ from $\mathcal{L}_{int}$ to $\widehat{\mathcal{L}_p}$ as follows:

$$\mathcal{T}r : \mathcal{L}_{int} \longrightarrow \widehat{\mathcal{L}_p}$$

where $\mathcal{T}r(\mathcal{A}) = \uparrow\mathcal{A} \vee \overrightarrow{\mathcal{A}} \vee \downarrow\mathcal{A}$.

With this function we have available "points versions" of the formulas of $\mathcal{L}_{int}$, and then we can treat events adequately in terms of points.

Now, we define the language $\mathcal{L}$ of $LN_{int}$ adding to the language $\widehat{\mathcal{L}_p}$ the following defined formulas:

$$\mathcal{A} =_{\text{def}} \mathcal{T}r(\mathcal{A}) \text{ for all formula } \mathcal{A} \in \mathcal{L}_{int}$$

We have in $\mathcal{L}$ the desired connective to achieve an absolute temporal treatment of events: if $\mathcal{A}$ is a well-formed formula in the sub-language $\mathcal{L}_{int}$, we define

$$\mathcal{A}\, at_I\, [\underline{m}, \underline{n}] =_{\text{def}} (\uparrow\mathcal{A} \wedge \downarrow\mathcal{A} \approx^{+} \underline{n})\ at\ \underline{m}$$

$\mathcal{A}\, at_I\, [\underline{m}, \underline{n}]$ reads *the event $\mathcal{A}$ occurs exactly in the interval* [m, n].

Once we have introduced the language, we define the semantics, which must confirm the readings of the connectives.

## 2.2 Topological Semantics of LNint

The semantics of LNint is defined considering $(T, +, \leq)$ as the time flow. It uses as key concepts the following:

For all wff $A$ and all time instants $t$,

$$m^{+}_{tA} = \min\{t' \in T \mid t' \text{ is an instant after } t \text{ such that } A \text{ is true at } t'\}$$

and

$$m^{-}_{tA} = \max\{t' \in T \mid t' \text{ is an instant before } t \text{ such that } A \text{ is true at } t'\}$$

we convey that $\min\emptyset = +\infty$, and that $\max\emptyset = -\infty$ and consider $\leq$ extended, in the usual way, to $T \cup \{+\infty\} \cup \{-\infty\}$.



We define a *temporal interpretation* for LNint, $h$, as any function that associates each atom with a subset of $T$:

$$h: \Omega^{I}_{p} \longrightarrow 2^{T}$$

and such that $h$ satisfies the followings requirements:

$h(\underline{t}) = \{t\}$, for all $\underline{t} \in \underline{T}$

$h(\bot) = \emptyset$

$h(\top) = T$

$h(\uparrow\alpha) \cap h(\downarrow\alpha) = h(\uparrow\alpha) \cap h(\overrightarrow{\alpha}) = h(\downarrow\alpha) \cap h(\overrightarrow{\alpha}) = \emptyset$

If $t \in h(\uparrow\alpha)$ then $(t, m^{+}_{t\downarrow\alpha}) \subseteq h(\overrightarrow{\alpha})$

If $t \in h(\downarrow\alpha)$ then $(t, m^{+}_{t\uparrow\alpha}) \cap h(\overrightarrow{\alpha}) = \emptyset$

where: $m^{+}_{t\mathcal{P}} = \min\big((t, +\infty) \cap h(\mathcal{P})\big)$, and $m^{-}_{t\mathcal{P}} = \max\big((-\infty, t) \cap h(\mathcal{P})\big)$.

The function $h$ can be extended to any wff of $\mathcal{L}$ as follows (we denote the wffs of $\mathcal{L}$ using the calligraphic letters $\mathcal{P}$, $\mathcal{Q}$, ...):

$$\begin{array}{rcl} h(\neg\mathcal{P}) & = & T \setminus h(\mathcal{P}) \\ h(\mathcal{P} \vee \mathcal{Q}) & = & h(\mathcal{P}) \cup h(\mathcal{Q}) \\ h(\mathcal{P} \wedge \mathcal{Q}) & = & h(\mathcal{P}) \cap h(\mathcal{Q}) \\ h(\mathcal{P} \rightarrow \mathcal{Q}) & = & (T \setminus h(\mathcal{P})) \cup h(\mathcal{Q}) \\ h(\mathcal{P} \leftrightarrow \mathcal{Q}) & = & h(\mathcal{P} \rightarrow \mathcal{Q} \wedge \mathcal{Q} \rightarrow \mathcal{P}) \\ h(\mathcal{P} \preccurlyeq \mathcal{Q}) & = & \{t \in T \mid m^{+}_{t\mathcal{P}} < +\infty \text{ and } m^{+}_{t\mathcal{P}} \leq m^{+}_{t\mathcal{Q}}\} \\ h(\mathcal{P} \succcurlyeq \mathcal{Q}) & = & \{t \in T \mid m^{-}_{t\mathcal{P}} > -\infty \text{ and } m^{-}_{t\mathcal{P}} \geq m^{-}_{t\mathcal{Q}}\} \end{array}$$

We denote with $H(\mathcal{A})$ the set of intervals $[t_1, t_2]$ at which $\mathcal{A}\ at_I\ [\underline{t_1}, \underline{t_2}]$ is true.

We can already give the interpretation of the future connectives which are defined in $\mathcal{L}_p$

$$\begin{array}{rcl} h(\oplus A) & = & \{t \in T \mid m^{-}_{tA} = t + 1\} \\ h(FA) & = & \{t \in T \mid m^{+}_{tA} < +\infty\} \\ h(GA) & = & \{t \in T \mid m^{-}_{t\neg A} = +\infty\} \\ h(A \prec B) & = & \{t \in T \mid m^{+}_{tA} < m^{+}_{tB}\} \\ h(A \approx^{+} B) & = & \{t \in T \mid m^{+}_{tA} < +\infty, m^{+}_{tA} = m^{+}_{tB}\} \end{array}$$

$$h(A\ at\ \underline{m}) = \begin{cases} T & \text{if } m \in h(A) \\ \emptyset & \text{otherwise} \end{cases}$$

$$h(\mathcal{A}\, at_I\, [\underline{m}, \underline{n}]) = \begin{cases} T & \text{if } [m, n] \in H(\mathcal{A}) \\ \emptyset & \text{otherwise} \end{cases}$$

Finally, we define that a formula $F$ of $\mathcal{L}$ is *valid in a temporal interpretation* $h$, denoted as $\models_h F$, if $h(A) = T$. A formula $F$ is called *valid*, denoted as $\models F$, if $\models_h F$ for all temporal interpretations $h$.

# 3 The Optimistic Method.

We will now introduce the basic ideas of the execution control environment of a database:

- Granule: indivisible unit of data from the point of view of concurrent-accesses control.
- Action: the basic treatment of a granule. According to the definition of granule, this will be the indivisible instruction of concurrent-accesses control.
- Transaction: set of actions produced by a particular input message; the execution of a specific program, which has the mandatory properties of definitiveness, indivisibility and constancy.
  - Definitiveness: changes produced on the data may only be undone by a contrary transaction; the effects of the original transaction have definitive character.
  - Indivisibility: all updates to the database must be completed and confirmed before the planned transaction has ended or before it is cancelled.
  - Constancy: when the planned transaction has been executed, the database maintains its original pre-updates properties whether or not it is confirmed.

To control the concurrent execution of transactions over a database we can use two kind of techniques: the pessimistic techniques and the optimistic techniques. The pessimistic ones suppose that the database has a high level of concurrency, hence there will be a lot of interferences among the execution of the transactions. The method imposes some constraints on execution to ensure the correctness and coherence of the information stored in the database. Unlike this technique, the other one will be used when the traffic of actions over the database be smaller than that in the case above or when the most of the transactions are *read-only* transactions. In this situation, the transaction will be executed with absolute liberty and when it is going to confirm its execution, the database manager system (DBMS) checks that it has not produced unwanted interferences.

The technique we will describe was presented in [9]. In the definition of this method two key concepts are used:

- *Read-Set* of the transaction $T$: is the set of granules which are used by $T$.
- *Write-Set* of the transaction $T$: is the set of granules which are updated by $T$.

The execution of the transaction $T$ will be separated into three phases:

- Working Phase: the transaction will be executed with absolute liberty and it will work on private copies of the granules that the DBMS gives to it.
- Checking Phase: the DBMS checks if there is any granule in the intersection of the read-set of $T$ and the write-set of any transaction which was checked when $T$ was working.
- Committing Phase: if the checking phase was successful, then the private copies of the granules become the official granules. If the check fails, then $T$ will be cancelled and it will have to be executed again.

To ensure the correctness of this method, the checking phase and the committing phase must constitute an *exclusive phase*: if one transaction is running its checking or committing phases, no other transaction can be either checking nor committing.



In [13] a refinement of this technique is presented: under some situations, the cancel of the transaction $T$ is not necessary, because the existence of a granule in the intersection of the read-set of $T$ and the write-set of other transaction does not imply that an interference has taken place. For example, if $T_2$ was checked while $T_1$ was working, but $T_2$ commits his updates to granules before $T_1$ read these granules, then $T_1$ reads a correct version of them. Nevertheless the database manager system cancels $T_1$ when it tries to check its execution.

Hence, we will not compare all the granules in the read-set of $T_1$ with the write-set of $T_2$, but a subset of the read-set of $T_1$ built up with the granules which were read *before* $T_2$ finished its committing phase.

# 4 Application of LNint.

We employ an intuitive approach to describe this transaction control method that uses our point-interval temporal logic. As we explained in the introduction it is very rewarding because:

- This logic can be applied as a program control language.
- The possibility of combining points and intervals is very appropriate for specifying the execution constraints over the transactions, because we must handle the phases of the transactions, that are naturally considered as events, and writing and reading actions, which have instantaneous executions.
- We can refer to the start and end points of events. Besides, we have a set of point temporal connectives with full expressive power [5], so we can express any real-world temporal situation which involves points.
- Because of the reasons above, our system can state intuitively, completely, and subtly all the situations that could occur in this field.

In our application it is necessary to consider first order LNint. The extension of propositional LNint to first order is made in the usual way [8].

To apply this logic, it is sufficient to specify the predicates to be valued and to construct a set of rules that forms a program that controls the execution of the different transactions.

PREDICATES:

The predicates we need are:

- Point predicates
  - $R(T,g)$ *Reading action over a granule g by T transaction*
  - $U(T,g)$ *Updating action over a granule g by T transaction*
  - $W(T,g)$ *Writing action over a granule g by T transaction*
- Event predicates
  - $\mathcal{W}(T)$ *Working phase of T transaction*
  - $\mathcal{V}(T)$ *Checking phase of T transaction*
  - $\mathcal{C}(T)$ *Committing phase of T transaction*

In general, in the paradigm *Declarative Past - Imperative Future* under which we are developing our application, it is necessary to distinguish two classes of predicates:

those that correspond to *environment facts* and those that correspond to *program facts*. The former collect the facts given by the user (in our case it is the transaction); i.e., facts that are not under the control of the program, The latter collect facts that are executed by the program (or controller).

In our particular case, the predicate $R(T,g)$ reflects an environment fact, while $\mathcal{W}(T)$ is a program fact.

Before the introduction of the rules, we enumerate a set of conventions adopted in our specification. The conventions are:

- The time instant with identified the time consumed for an action.
- We assume the Checking Phase consumes only one instant (there is not really a restriction, it is only an assumption to simplify the specification).
- We consider the Committing Phase always takes place. In the case of a cancelled transaction, we assume that the commitment has duration 1; i.e., $\uparrow \mathcal{C}(T)$ is followed for $\downarrow \mathcal{C}(T)$ and $\mathcal{C}(T)$ has not "$\overrightarrow{\mathcal{C}(T)}$ points".

RULES:

The set of rules is:

**I. Rules of description (ordering of transaction phases):**

- $\mathcal{W}(T) \rightarrow \mathcal{W}(T) \wedge \langle ab^{+} \rangle \mathcal{V}(T)$
- $\mathcal{V}(T) \rightarrow \mathcal{V}(T) \wedge \langle ab^{+} \rangle \mathcal{C}(T)$

  The working phase is always followed by a checking phase and this one by a committing phase.

- $\mathcal{C}(T) \rightarrow \mathcal{C}(T) \wedge \langle ab^{-} \rangle \mathcal{V}(T)$
- $\mathcal{V}(T) \rightarrow \mathcal{V}(T) \wedge \langle ab^{-} \rangle \mathcal{W}(T)$

  The committing phase is always preceded by a checking phase and this one by a working phase.

- $\neg\Big(\mathcal{W}(T) \wedge \langle over^{-} \rangle (\mathcal{V}(T) \vee \mathcal{W}(T))\Big)$
- $\neg\Big(\mathcal{V}(T) \wedge \langle over^{-} \rangle \mathcal{C}(T)\Big)$

  The phases of one transaction cannot overlap

**II. Rules for the definition of the critical section:**

- $\mathcal{V}(T_1) \rightarrow \mathcal{V}(T_1) \wedge \neg\langle over^{-} \rangle \mathcal{V}(T_2) \wedge \neg\langle dur \rangle \mathcal{V}(T_2) \wedge$
  $\wedge \neg\langle beg \rangle \mathcal{V}(T_2) \wedge \neg\langle end \rangle \mathcal{V}(T_2)$

  The checking phase of a transaction cannot be executed simultaneously with the checking phase of another transaction

- $\mathcal{V}(T_1) \rightarrow \mathcal{V}(T_1) \wedge \neg\langle over^{-} \rangle \mathcal{C}(T_2) \wedge \neg\langle dur \rangle \mathcal{C}(T_2) \wedge$
  $\wedge \neg\langle beg \rangle \mathcal{C}(T_2) \wedge \neg\langle end \rangle \mathcal{C}(T_2)$



The checking phase of a transaction cannot be executed simultaneously with the committing phase of another transaction

- $\mathcal{C}(T_1) \rightarrow \mathcal{C}(T_1) \wedge \neg\langle over^{-} \rangle \mathcal{C}(T_2) \wedge \neg\langle dur \rangle \mathcal{C}(T_2) \wedge$
  $\wedge \neg\langle beg \rangle \mathcal{C}(T_2)) \wedge \neg\langle end \rangle \mathcal{C}(T_2)$

  The committing phase of a transaction cannot be executed simultaneously with the committing phase of another transaction

- $\mathcal{C}(T_1) \rightarrow \mathcal{C}(T_1) \wedge \neg\langle over^{-} \rangle \mathcal{V}(T_2) \wedge \neg\langle dur \rangle \mathcal{V}(T_2) \wedge$
  $\wedge \neg\langle beg \rangle \mathcal{V}(T_2)) \wedge \neg\langle end \rangle \mathcal{V}(T_2)$

  The committing phase of a transaction cannot be executed simultaneously with the checking phase of another transaction

**III. Rules for the definition of the working phase of a transaction:**

- $R(T,g) \rightarrow \overrightarrow{\mathcal{W}(T)}$
- $U(T,g) \rightarrow \overrightarrow{\mathcal{W}(T)}$

  The reading and updating actions always take place during the working phase

**IV. Specification of the method:**

- $\overrightarrow{\mathcal{V}(T_1)} \wedge Interf(T_1,T_2) \rightarrow Cancel(T_1)$
- $\overrightarrow{\mathcal{V}(T_1)} \wedge \neg Interf(T_1,T_2) \rightarrow Cont(T_1)$

  If an interference occurs in the checking phase of a transaction, it is cancelled; otherwise, it continues its execution.

- $\overrightarrow{\mathcal{C}(T)} \wedge PendingWriting(T) \rightarrow Write(T)$
- $\overrightarrow{\mathcal{C}(T)} \wedge \neg PendingWriting(T) \rightarrow End(T)$

  During the committing phase of a transaction the existence of pending writing actions is examined. If any exists, then the operation is executed otherwise the transaction is finished.

Where:

- $Interf(T_1,T_2) =_{def} \Big(U(T_1,g) \succ \uparrow\mathcal{W}(T_1) \vee$
  $R(T_1,g) \succ \uparrow\mathcal{W}(T_1)\Big) \wedge$ (a)
  $U(T_2,g) \succ \uparrow\mathcal{W}(T_2) \wedge$ (b)
  $\downarrow\mathcal{C}(T_2) \succ \uparrow\mathcal{W}(T_1) \wedge$ (c)
  $\ominus\overrightarrow{\mathcal{C}(T_2)}\ atlast\ \downarrow\mathcal{C}(T_2)$ (d)

(a) and (b) reflect the existence of two conflicting actions over the same granule (c) reflects the simultaneity of the transactions and (d) eliminates interferences with cancelled transactions.

To avoid unnecessary cancellations following the refinement of [13] (as we mentioned in section 3) we must change (c) for

$$\downarrow\mathcal{C}(T_2) \succ (R(T_1, g) \vee U(T_1, g))$$

- $Cancel(T) =_{\text{def}} \oplus\Big( \downarrow\mathcal{V}(T) \wedge \oplus(\downarrow\mathcal{C}(T) \wedge \uparrow\mathcal{W}(T))\Big)$

  The checking phase is finished and the committing phase has no actions to execute.

- $Cont(T) =_{\text{def}} \oplus(\uparrow\mathcal{C}(T))$

  It is ordered the start of the committing phase of the transaction.

- $PendingWriting(T) =_{\text{def}} U(T, g) \succ \Big( \uparrow\mathcal{W}(T) \wedge W(T, g)\Big)$

  There is an updated granule that has not been written on secondary storage.

- $Write(T) =_{\text{def}} \oplus(W(T, g))$

  The writing operation over the granule corresponding to the pending updating action is executed.

- $End(T) =_{\text{def}} \oplus(\downarrow\mathcal{C}(T))$

  The transaction execution ends.

# References


[1] James A. Allen and Henry A. Kautz. A model of naive temporal reasoning. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of Commonsense World*, pages 251-268. Ablex Publishing Corporation, 1988.

[2] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, July 1984.

[3] Fahiem Bacchus, Josh Tenenberg, and Johannes A. Koomen. A non-reified temporal logic. *Artificial Intelligence*, 52:87–108, 1991.

[4] Alexander Bochman. Concerted instant-interval semantics i: Temporal ontologies. *Notre Dame Journal of Formal Logic*, 31(3):403–414, 1990.

[5] Alfredo Burrieza and Inmaculada Pérez de Guzmán. A new algebraic semantic approach and some adequate connectives for computation with temporal logic over discrete time. *Journal of Applied Non-Classical Logic*, 2, 1992.





[6] Thomas L. Dean and Drew V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[7] Manuel Enciso and Inmaculada Pérez de Guzmán. Controlando el tiempo continuo. In Manuel Hermenegildo and Juan J. Moreno, editors, *Primer Congreso Nacional de Programación Declarativa, PRODE'92*, 1992.

[8] D. M. Gabbay. *The declarative past and imperative future*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer Verlag, 1989.

[9] H. T. Kung and J.T. Robinson. On optimistic methods for concurrency control. In *Fifth International Conference on Very Large Database*, 1979.

[10] Drew V. McDermott. Nonmonotonic logic ii: Nonmonotonic modal theories. *Journal of the ACM*, 29(1), 1982.

[11] Ben C. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986.

[12] Inmaculada Pérez de Guzmán and Carlos Rossi. Lnint: a temporal logic that combines points and intervals and the absolute and relative approaches. Submitted to a technical journal, 1993.

[13] U. Pradel et al. Redesign of optimistic methods: Improving performance and applicability. In *International Conference of Data Engineering*, 1986.

[14] Yoav Shoham. *Reasoning about Change. Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, 1988.

[15] J. F. A. K. van Benthem. *The Logic of Time*. D. Reidel Publishing Company, 1983.

[16] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In *Readings on Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufmann, 1990.

# A Modal Extension of Logic Programming

Matteo Baldoni, Laura Giordano and Alberto Martelli
Dipartimento di Informatica - Università di Torino
C.so Svizzera 185 - 10149 TORINO
E-mail: {baldoni,laura,mrt}@di.unito.it

**Abstract**

In this paper we present a modal extension of logic programming, which provides reasoning capabilities in a multiagent situation. The language contains embedded implications, modal operators $[a_i]$ to represent agent beliefs, together with a kind of "common knowledge" operator. In this language we can also define modules, compose them in several ways, and, also, we can perform hypothetical reasoning.

## 1 Introduction

The problem of extending logic programming languages with modal operators, has been studied by several researchers. In particular, in [4] an extension of Prolog with modal operators, called MOLOG, is proposed, and a resolution procedure, close to Prolog resolution, is defined for modal Horn clauses in the logic S5 which contain universal modal operators of the form Know(a). Another modal extension of logic programming is the temporal logic programming language TEMPLOG introduced in [1]. Moreover, in [20] a logic language extended with a modal operator *assume* is defined, which allows atomic updates to be performed on the program.

Recently we have studied how structuring facilities, like blocks and modules, can be introduced in logic programming languages by making use of modal extensions [9, 2]. In particular, in [9] we have focused on a language with *embedded implications*, like the languages in [7, 6, 14, 12, 8]. These languages allow implications of the form $D \rightarrow G$ to occur both in goals and in clause bodies, and this provides a way of introducing local definitions of clauses: the clauses in $D$ are intended to be local to the goal $G$, as they can be used only in a proof of $G$. The meaning of hypothetical implications, is that of hypothetical insertion: the goal $D \rightarrow G$ is derivable from a program $P$ if $G$ is derivable from a new program updated with $D$. When intuitionistic logic is taken as the underlying logic of this language, like in N_Prolog and in $\lambda$-Prolog , embedded implications allow hypothetical reasoning to be performed.

In [8] we have shown how different languages with embedded implications can be obtained by choosing different visibility rules for locally defined clauses. A modal extension of Horn clause logic (based on S4 logic) can provide a unifying framework in



which these different kinds of local definitions of clauses can be defined and integrated [9]. This extension provide a notion of block, from which various kinds of modules can also be defined, by introducing some syntactic sugar.

In [2] we have proposed a modal extension of Horn clause logic which provides modules as a basic feature. This extension is defined on the line of the above mentioned language with blocks, though in this case the language does not contain embedded implications. On the contrary, modules are defined by introducing different modal operators ($[m_1], \ldots, [m_k]$) each one associated with a module. Module composition can be obtained by allowing modules to export clauses or derived facts. To achieve this purpose, a different modal operator $\Box$ is introduced, which makes possible to distinguish clauses local to a module from those that are fully exported and those whose consequences are exported. This language allows one to model different kinds of modules presented in the literature (so that in each situation the kinds of module that suit better can be adopted); furthermore, this language provides some well known features of object oriented programming, like the possibility of representing dependencies among modules in a hierarchy, and the notion of *self* to reason on this hierarchy.

Although these extensions were introduced with the aim of defining modules, the modal operators introduced in this language can be given alternative meanings, which can allow a more general use of the language. For instance, the language can be suitable for representing knowledge and belief by considering the modal operators $[m_i]$ as "belief" operators, and the $\Box$ operator as a kind of "common knowledge" operator. The aim of this paper is to integrate all previously mentioned extensions (modal operators and hypothetical implications) in an extended language to be used for knowledge representation. The language should be as general as possible, while retaining the distictive feature of logic programming of having a goal directed operational semantics.

The logic programming language presented in this abstract is a modal logic refinement of hereditary Harrop formulas [17], and it lies on the same line as other logic programming languages which are not based on classical first-order logic, like those based on intuitionistic logic [7, 6, 12, 13, 14, 16, 15] higher-order logic [17] and linear logic [11]. This language we define subsumes the languages defined in [9, 2], so that it allows both to define module constructs and to perform hypothetical reasoning. Moreover it provides a simple way to formulate reasoning capabilities in a multiple agent situation.

## 2 The Language

In this section we introduce a modal logic programming language which subsumes the languages defined in [9, 2], so that it allows both to define module constructs and to perform hypothetical reasoning. Moreover it provides a simple way to formulate reasoning capabilities in a multiple agent situation. We extend Horn clause language, with $k$ modal operators $[a_1], \ldots, [a_k]$, where the $a_i$'s are constants, each one representing an agent, and a modal operator $\Box$, which is a sort of common knowledge operator. Each modal formula $[a_i]\alpha$ can be read "agent i believes $\alpha$".

Let $A$ be an atomic formula and $T$ a distinguished proposition (true). The syntax of the language is the following:

$$G ::= T \mid A \mid G_1 \wedge G_2 \mid \exists x G \mid [a_i]G \mid \Box G \mid [a_i](D \supset G) \mid \Box(D \supset G)$$
$$D ::= G \supset H \mid [a_i]D \mid \Box D \mid \forall x D$$
$$H ::= A \mid [a_i]H \mid \Box H ,$$

where $G$ stands for a goal, $D$ for a clause, and $H$ for a clause head. In the following D will be interchangeably regarded as a conjunction and a set of clauses. A program $P$ consists of a set of clauses $D$.

In this language, modal operators $[a_i]$ and $\Box$ can freely occur in front of clauses, in front of clause heads, in front of each goal and, in particular, in front of embedded implications (or implication goals). $[a_i]G$ is a goal and means that G has to be proved in the beliefs of agent $i$ while $[a_i]D$ means that the clause D is part of the beliefs of agent $i$. As regards implication goals, they must be preceded by at least one modal operator. Note that the language is extended with respect to the one presented in [2], where the definition of clauses were restricted to few different shapes (in particular, arbitrary sequences of modal operators were not admitted) and embedded implications were not allowed.

As an example consider the following formulation of the *(two) wise men puzzle*:

(1) $\Box\ (bs(b) \supset ws(a))$

(2) $\Box(bs(b) \supset [a]bs(b))$

(3) $\Box([b]((T \supset bs(b)) \supset \bot) \supset [b]ws(b))$

(4) $[b]([a]ws(a) \supset \bot)$

In this example, $\bot$ is a distinguished proposition representing falsity; $bs(a)$ ($ws(a)$) represents the fact that $a$ has a black (white) spot on his forehead. [a] is a modal operator and $[a]F$ means that agent $a$ knows $F$. All the clauses preceded by the modal operator $\Box$, denote the information which is common to all agents. In clause (3), a form of negation has been introduced as usually in a language with embedded implications, by making use of the proposition $\bot$: *"not F"* is expressed by the implication $F \supset \bot$. So clause (3) can be read: if agent $b$ knows that he has not the black spot, then he knows that he has the white spot. (4) says that $b$ knows that $a$ doesn't know if he has the white spot on his forehead. This models the situation in which $a$ is the first which is asked, he doesn't know the answer and $b$ knows that. In this example, while the modal operators $[a]$, $[b]$ and $\Box$ give a way to distinguish among information of the single agents and information common to all of them, embedded implications allow forms of hypothetical reasoning to be performed.

Following [2], in our language the modal operators $[a_i]$ are all ruled by the axioms of K, and, in particular,

$$[a_i](B \supset A) \supset ([a_i]B \supset [a_i]A)$$

is part of the axiomatization, for each $[a_i]$. On the other hand, the operator $\Box$ is ruled by the axioms of S4. Thus the axiomatization contains the following axioms:



(K) $\Box(B \supset A) \supset (\Box B \supset \Box A)$
(T) $\Box A \supset A$
(4) $\Box A \supset \Box\Box A$.

Moreover, since the operator $\Box$ is intended to represents what is known by all the agents, it interacts with each operator $[a_i]$ by the following interaction axioms,

$$\Box A \supset [a_i]\Box A.$$

This language is quite similar to the modal language introduced in [10] for dealing with the notions of knowledge and common knowledge, though, the modal operator $\Box$ we have introduced does not exactly coincide with (and it is weaker than) the common knowledge operator in [10]. In particular, it holds that, for all modal operators $[a_i]$, $\Box\alpha \supset [a_i]\Box\alpha$, so that if $\Box\alpha$ holds, then all the agents believe $\Box\alpha$. However, it does not hold that $[a_1]\Box A \wedge \ldots \wedge [a_k]\Box A \wedge A \supset \Box A$ while it is expected to hold when $\Box$ is the common knowledge operator. In [2] we have defined both Kripke semantics and a sequent calculus for a modal language $L$ containing the logical conectives $\neg$, $\wedge$, $\supset$, $\forall$ and $\exists$, and the modal operators $[a_1], \ldots, [a_k]$ and $\Box$. In the next section, we will recall the sequent calculus.

## 3 Sequent calculus

For simplicity it has been assumed that the language $L$ does not contain function symbols, but only constants. Let's assume that the language $L$ contains countably many constants, variables and relational symbols. Since, as mentioned above, the modal operator $\Box$ is of type S4, we present a cut-free sequent calculus for the language $L$ which extends the cut-free sequent calculus for S4 presented in [19] (section 2.1) and adapted from [5]. In addition to the rules of the calculus for S4, a new rule is needed to deal with each modal operator $[a_i]$.

As a difference with the calculus in [19], and also with the calculus we have proposed in [2], we will not include the rules for negation ($L\neg$) and ($R\neg$), since this not the kind of negation we want in our language. Here, we only consider the positive fragment of the language in [2]. As mentioned above, we introduce a form of negation in the language, by making use of implication and of the distingiushed symbol $\bot$, representing falsity. As in minimal logic, $\bot$ is not required to satisfy any particular property. The calculus is the following:

$$\frac{}{\Gamma, A \rightarrow A, \Delta}$$

$$\frac{\Gamma, A, B \rightarrow \Delta}{\Gamma, A \wedge B \rightarrow \Delta}(L\wedge) \qquad \frac{\Gamma \rightarrow B, \Delta \quad \Gamma \rightarrow C, \Delta}{\Gamma \rightarrow B \wedge C, \Delta}(R\wedge)$$

$$\frac{\Gamma \rightarrow A, \Delta \quad \Gamma, B \rightarrow \Delta}{\Gamma, A \supset B \rightarrow \Delta}(L\supset) \qquad \frac{\Gamma, A \rightarrow B, \Delta}{\Gamma \rightarrow A \supset B, \Delta}(R\supset)$$

$$\frac{\Gamma, [x/c]A \to \Delta}{\Gamma, \forall x A \to \Delta}(L\forall) \qquad \frac{\Gamma \to [x/a]A, \Delta}{\Gamma \to \forall x A, \Delta}(R\forall)$$

$$\frac{\Gamma, [x/a]A \to \Delta}{\Gamma, \exists x A \to \Delta}(L\exists) \qquad \frac{\Gamma \to [x/c]A, \Delta}{\Gamma \to \exists x A, \Delta}(R\exists)$$

$$\frac{\Gamma, A \to \Delta}{\Gamma, \Box A \to \Delta}(L\Box) \qquad \frac{\Gamma^* \to A}{\Gamma \to \Box A, \Delta}(R\Box)$$

$$\frac{\Gamma^*, \Gamma_i^* \to A}{\Gamma \to [a_i]A, \Delta}(R[a_i])$$

where $\Gamma^* = \{\Box\alpha : \Box\alpha \in \Gamma\}$ and $\Gamma_i^* = \{\alpha : [a_i]\alpha \in \Gamma\}$. For $(R\forall)$ and $(L\exists)$ here is the proviso that $a$ is a parameter that does not occur in any formula of the lower sequent. In rule $(L\forall)$ and $(R\exists)$ $c$ is any constant of the language.

In this sequent calculus there is no need for structural rules, since in a sequent $\Gamma \to \Delta$ the antecedent and the succedent are sets of statements rather than sequences of statements.

Since $T$ is a distinguished symbol which can be regarded as any propositional tautology, we can assume to have the additional initial sequent $\Gamma \to T, \Delta$ to deal with this symbol.

A proof for the sequent $\Gamma \to \Delta$ is a finite tree constructed using the above rules, having the root labelled with $\Gamma \to \Delta$ and the leaves labelled with initial sequents, i.e. sequents of the form $\Gamma, A \to A, \Delta$ or of the form $\Gamma \to T, \Delta$.

Note that the rules $(L\Box)$ and $(R\Box)$ above are exactly the same as those of the calculus for S4 in [19]. The only difference is due to the fact that in our language L the existential modal operator $\Diamond$ is not present. The inference rule $(L\Box)$ is needed since the modal operator $\Box$ is of type $S4$, and, in the Kripke semantics, the accessibility relation associated with it is reflexive.

The inference rule $(R[a_i])$ is needed to deal with the modal operator $[a_i]$, and it is quite similar in style to the rule $(R\Box)$. As a difference with $(R\Box)$, the antecedent of its premise contains both the set $\Gamma^*$ and the set $\Gamma_i^*$ . This corresponds to the fact that all the formulas of type $\Box\alpha$ are visible from within the context of a modal operator $[a_i]$. Moreover, since the modal operators $[a_i]$ are of type $K$, and, in the Kripke semantics, the accessibility relation associated with each of them is not reflexive, there is no rule $(L[a_i])$.

# 4 A goal directed proof procedure

A goal directed proof procedure can be defined for this language with modal operators. Following [16], in order to avoid problems with variable renaming and substitutions, given a set of clauses $D$, we denote by $[D]$ the set of all ground instances of the clauses in $D$. We introduce a notion of operational derivability of a closed goal $G$ from a program $P$ by induction on the structure of G. Since the language allows



modal operators, in the operational semantics we will introduce a notion of operational derivability of a closed goal $G$ from a program $P$ in a certain modal context. A modal context is a sequence of modal operators $L_1|...|L_n$, which represents the sequence of right rules $((R\Box)$ or $(R[a_i]))$ that have been applied in the corresponding sequent proof, up to that point. To deal with embedded implications, each modal operator $L_j$ in the sequence is associated with a (possibly empty) set of clauses $D_j$.

Hence, we define the operational derivability of a closed goal $G$ from a list of pairs

$$(L_0, D_0) \mid (L_1, D_1) \mid \ldots, (L_n, D_n)$$

(where, for uniformity, $D_0$ is the initial program $P$, and $L_0$ is not used), by induction on the structure of $G$, as follows (we will denote by $\Gamma$ an arbitrary sequence of modal operators):

1. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash T$;
2. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash A$ if for some $j = 0, \ldots, n$, there is a clause $\Gamma_b(G \supset \Gamma_h A) \in [D_j]$ and a $k$, $j \leq k \leq n$, such that
   $\Gamma_b$ matches $\Gamma_1 = L_{j+1} \ldots L_k$,
   $\Gamma_h$ matches $\Gamma_2 = L_{k+1} \ldots L_n$, and
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_k, D_k) \rangle \vdash G$;
3. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash G_1 \wedge G_2$ if
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash G_1$
   and $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash G_2$;
4. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash L_{n+1}G$ if
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \mid (L_{n+1}, \emptyset) \rangle \vdash G$;
5. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash L_{n+1}(D \supset G)$ if
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \mid (L_{n+1}, D) \rangle \vdash G$;
6. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash \exists x G$ if, for some closed term t,
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash [x/t]G$.

In the definition above, the rules for the conjunction and existentially quantified goals are the usual ones. A goal $[a_i]G$ is proved by adding the modal operator $[a_i]$ to the current context and proving $G$ in the resulting context. The same holds for the modal operator $\Box$.

Embedded implications are always preceded by a modal operator. To prove a goal $L_{n+1}(D \supset G)$, where $L_{n+1}$ is an arbitrary modal operator, the modal operator $L_{n+1}$ is added to the current context together with the clauses in $D$, and $G$ is proved in the resulting context.

To prove an atomic goal $A$ a clause is selected in the initial program $D_0$ or from a set of clauses $D_j$ that has been added in the context, by proving some implication goal. In both cases, to verify that the clause is applicable in the current context, it must be checked whether the modal operators in the clause (both in front of it and in front of its head) match the current context, from $L_{j+1}$ to $L_n$. In particular, the sequence of the modal operators $\Gamma_b$ in front of the selected clause must match a prefix of the sequence $L_{j+1} \ldots L_n$, while the sequence of the modal operators $\Gamma_h$ in front of the head of the selected clause must match the remaining part of the sequence. We say that a sequence $\Gamma$ of modal operators matches another sequence $\Gamma'$ if each modal operator in $\Gamma$ matches a sequence of operators in $\Gamma'$ in the ordering. More precisely, each modal operator $[a_i]$ in $\Gamma$ matches the operator $[a_i]$ itself, while each operator $\Box$ in $\Gamma$ may match an arbitrary (possibly empty) sequence of modal operators in $\Gamma'$.

Formally, a sequence $\Gamma = L_1 \ldots L_n$ of modal operators matches another sequence $\Gamma'$ if each modal operator $L_j$ in $\Gamma$ can be associated with a sequence of modal operators $f_j(L_j)$ such that the following condition hold:

- $f_j([a_i]) = [a_i]$, for all modal operators $[a_i]$;
- $f_j(\Box)$ is any sequence (including the empty sequence of modal operators);
- $f_1(L_1) \ldots f_n(L_n) = \Gamma'$.

Given a program $P$ and a goal $G$, we say that $G$ is provable from $P$, if

$$\langle (L_0, P) \rangle \vdash G$$

can be derived by applying the operational rules above. It is possible to prove that the above operational semantics is sound and complete with respect to the sequent calculus above, i.e., $\langle (L_0, P) \rangle \vdash G$ iff the sequent $P \rightarrow G$ is provable.

# 5 Examples

**Example 1** We give a formulation of the *(three) wise men puzzle*. The formulation is quite similar to the one of the two wise man puzzle in section 2. However, in order to avoid introducing many variant of the same clause for the different agents, we make use of parametric modal operators as a shorthand.

(1) $\Box\ \forall X, Y, Z(bs(X) \wedge bs(Y) \wedge X \neq Y \wedge X \neq Z \wedge Y \neq Z \supset ws(Z))$
(2) $\Box \forall X, W(bs(X) \wedge X \neq W \supset [W]bs(X))$
(3) $\Box \forall X([X]((T \supset bs(X)) \supset \bot) \supset [X]ws(X))$
(4) $\Box([a]ws(a) \supset \bot)$
(5) $[c]([b]ws(b) \supset \bot)$

We have modelled the situation in which $a$, $b$ and $c$ are asked in this ordering. $a$ doesn't know if he has the white spot on his forehead; $b$ doesn't know too; $c$ knows about their answers (clauses (4) and (5)) and, hence, he can conclude that he has the white spot himself.

The goal $G = [c]ws(c)$ succeeds from the program, with the following derivation:



$\langle (L_0, P) \rangle \vdash [c]ws(c)$
$\langle (L_0, P) \mid ([c], \emptyset) \rangle \vdash ws(c)$
$\langle (L_0, P) \rangle \vdash [c]((T \supset bs(c)) \supset \bot)$, by clause (3) with $X = c$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash \bot$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash [b]ws(b)$, by clause (5)
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], \emptyset) \rangle \vdash ws(b)$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash [b]((T \supset bs(b)) \supset \bot)$,
by clause (3) with $X = b$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash \bot$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash [a]ws(a)$, by clause (4)
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash ws(a)$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash$
$bs(X) \wedge bs(Y) \wedge X \neq Y \wedge X \neq a \wedge Y \neq a$ by clause (1)

Let us consider separately the subqueries $bs(X)$. We have the following subderivations:

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash bs(X)$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash bs(X)$,
by clause (2) with $W = a$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash T$, by taking $X = b$

and similarly:

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash bs(Y)$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash bs(Y)$,
by clause (2) with $W = a$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash bs(Y)$, by clause (2) with $W = b$
$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash T$, by taking $Y = c$

Hence, $bs(X) \wedge bs(Y)$ succeeds with $X = b$ and $Y = c$. Thus, $X \neq Y \wedge X \neq a \wedge Y \neq a$ succeeds too. And this concludes the proof.

**Example 2** As mentioned above, the modal language we have defined in this paper subsumes the language proposed in [2] to introduce module constructs in Horn clause logic. In particular, the language in [2] associates a modal operator $[m_i]$ with each module and it provides module definitions of the form $\Box[m_i]D$ (where $D$ is a set of clauses), with the meaning that the clauses in $D$ belong to module $[m_i]$. The operator $\Box$ in front of the module definition make it visible from inside other modules. The modal operators $[m_i]$ may occur in front of goals: $[m_i]G$ is a goal and it means

that $G$ has to be proved in the module $[m_i]$. This provides a simple way to define a flat collection of modules and to specify the proof of a goal in a module. In a simplistic view, modules are closed environments. However, different forms of module composition can be obtained by making use of the modal operator $\Box$ to distinguish among clauses local to a module and clauses exported by a module.

Consider for instance the query $[m_1][m_2][m_3]G$. the goal $G$ must be proved in the composition of modules $[m_1]$, $[m_2]$ and $[m_3]$. When modules are regarded as being open, each module may export information to the modules following it in the sequence. The language in [2] makes a distintion among clauses that are *local* to the module in which they are defined, $G \supset A$, clauses that are wholly exported by the module, $\Box(G \supset A)$, and clauses which only export their head, $G \supset \Box A$. Notice that the syntax we have defined in section 2, gives much more freedom, since an arbitrary sequence of modal operators may occur in front of a clause and its head.

Let us consider the following example (taken from [3]), describing inheritance in a hierarchy of modules:

$\Box[animal]$

$\{T \supset \Box mode(walk)$

$\Box(no\_of\_legs(2) \supset mode(run))$

$\Box(no\_of\_legs(4) \supset mode(gallop))\ \}$

$\Box[bird]$

$\{T \supset \Box no\_of\_legs(2)$

$T \supset \Box covering(feather)\ \}$

$\Box[tweety]$

$\{T \supset owner(fred)\ \}.$

The goal

$$[animal][bird][tweety]mode(run)$$

succeeds, since the clause defining $mode(run)$ is exported by the module $[animal]$ and its body can be evaluated in the current context, inculding module $[bird]$ which contains the information $no\_of\_legs(2)$. The goal would have failed, if the alternative clause $no\_of\_legs(2) \supset mode(run)$ had been contained in module $[animal]$. By using clauses preceded by the operator $\Box$ we can achieve a result quite similar to the use of *self* in object-oriented languages.

The more extended language we have defined in this paper, offers some additional features in defining modules. The possibility of using embedded implications in the language can be usefully exploited to specify module interfaces. Indeed, embedded implications can be used to hide predicates of a module that we do not want to export. For instance, let $m$ be a module containing a set of clause definitions

$G_1 \supset p_1$

$\ldots$

$G_n \supset p_n$



together with a set of clauses $D = \{D_1, \ldots, D_k\}$ that must be visible only from within the module $m$. We can define $m$ as follows:

$\Box[m]$

$\{(\Box(D \supset G_1) \supset p_1)$

$\ldots$

$(\Box(D \supset G_n) \supset p_n)\}.$

Embedded implications can also be used to couple inheritance and hypothetical reasoning, as done in [3].

Since in this modal language clauses can be preceded by an arbitrary sequence of modal operators, we can generalize module definitions $\Box[m_i]D$ above, by allowing *nested* module definitions as follows:

$$\Box[m_i]\Box[m_j]D,$$

where the module $m_j$ is defined locally to $m_i$, and it becomes visible whenever $m_i$ is entered.

**Example 3** We present the Fibonacci example from [1]. We use a modal operator $[next]$ to represent the next instant of time. We want $fib(x)$ to hold after $n$ instants of time, if $x$ is the n-th Fibonacci number. The formulation is the following:

1. $T \supset fib(0)$
2. $T \supset [next]fib(1)$
3. $\Box(fib(Y) \wedge [next]fib(Z) \wedge X \textbf{ is } Y + Z \supset [next][next]fib(X))$.

Clause (1) says that at time 0, $fib(0)$ holds; clause (2) says that at time 1, $fib(1)$ holds; clause (3) says that, for any time $n$, if $fib(Y)$ holds at time $n$, and if $fib(Z)$ holds at time $n+1$, then $fib(X)$, with $X = Y + Z$, holds at time $n+2$.

From this program, the query

$$[next][next][next]fib(X)$$

succeeds with $X = 2$, and indeed 2 is the 3-rd Fibonacci number.

## 6 Conclusions

In this paper we have defined a modal language which extends the languages proposed in [9, 2] to deal with blocks and modules. This language provides a simple way to formulate reasoning problems in a multiagent situation and it also provides hypothetical reasoning capabilities.

The language we have focused on contains universal modal operators $[a_i]$ of type $K$, and a modal operator $\Box$ which is a sort of common knowledge operator. Similar but

different languages could be defined by changing the properties of the modal operators $[a_i]$. For instance, the modal operators $[a_i]$ could follow the rules of $K4$ or $S4$, instead of the rules of $K$. In this cases it could still be possible to define an operational top down semantics similar to the one presented in section 4, by modifying the notion of *matching* between sequences of modal operators. A study of goal directed proof methods in the logic K, K4, S4, and also for S5, has been done in [18] for the fragment of strict implication.

## Acknowledgement

This work has been partially supported by CNR - Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo" under grant n. 92.01577.PF69.

## References

[1] M. Abadi and Z. Manna. Temporal logic programming. *J.Symbolic Computation*, (8):277–295, 1989.

[2] M. Baldoni, L.Giordano, and A.Martelli. A multimodal logic to define modules in logic programming. In *Proc. 1993 International Logic Programming Symposium*, pages 473–487, Vancouver, 1993.

[3] Antonio Brogi, Evelina Lamma, and Paola Mello. Inherritance and hypothetical reasoning in logic programming. In *Proc. European Conference on Artificial Intelligence*, pages 105–110, Stockholm, 1990.

[4] L. Fariñas del Cerro. Molog: A system that extends Prolog with modal logic. *New Generation Computing*, (4):35–50, 1986.

[5] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese library*. D. Reidel, Dordrecht, Holland, 1983.

[6] D. M. Gabbay. NProlog: An extension of Prolog with hypothetical implications.ii. *J.Logic Programming*, 2(4):251–283, 1985.

[7] D. M. Gabbay and N. Reyle. NProlog: An extension of Prolog with hypothetical implications.i. *Journal of Logic Programming*, (4):319–355, 1984.

[8] L. Giordano, A. Martelli, and G.F. Rossi. Extending Horn clause logic with implication goals. *Theoretical Computer Science*, 95:43–74, 1992.

[9] Laura Giordano and Alberto Martelli. A modal reconstruction of blocks and modules in logic programming. In *Proc. 1991 Int. Logic Programming Symposium*, pages 239–253, San Diego, October 1991.

[10] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.



[11] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In G. Kahn, editor, *Sixth Annual Symposium on Logic in Computer Science*, pages 32–42, Amsterdam, 1991.

[12] L. T. Mc Carty. Clausal intuitionistic logic. i. fixed-point semantics. *J. Logic Programming*, 5(1):1–31, 1988.

[13] L. T. Mc Carty. Clausal intuitionistic logic.ii.Tableau proof procedure. *J. Logic Programming*, 5(2):93–132, 1988.

[14] D. Miller. A theory of modules for logic programming. In *Proc. IEEE Symp. on Logic Programming*, pages 106–114, September 1986.

[15] D. Miller. Lexical scoping as universal quantification. In *Proc. 6th Int. Conf. on Logic Programming*, pages 268–283, Lisbon, 1989.

[16] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, (6):79–108, 1989.

[17] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as foundations for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[18] N. Olivetti and D. Gabbay. Goal-directed method for strict implication. Technical report, 1993.

[19] L. A. Wallen. *Automated Deduction in Nonclassical Logics*. The MIT Press, 1990.

[20] D. S. Warren. Database updates in pure Prolog. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, pages 244–253, Tokyo, 1984.