Syntax of **Milord II**

February 27, 1996
Version 5.0
IIIA-CSIC

1.1 Notation

The symbols ::=, [,], | are part of the BNF formalism, as follows:

L ::= R	The syntax of L is defined by R
[X]	An optional item
X | Y	An item from one of the syntactic categories X or Y

- We write the predefined terminal symbols that are part of the language **Milord II** in <u>**underlined boldface**</u> [1].
- We write user-defined terminal symbols in *italic*. They are always atomic symbols.
- We write non-terminal symbols in normal type face.

Lines of comments can be written after two semicolons (;;). If the comment is larger that one line, two semicolons must be written at the beginning of the lines. Several spaces and carry returns are ignored and considered only a space.

[1] Comparison of predefined terminal symbols is case-insensitive.

1.2 Modular System

environ	::=	moddecl environ | moddecl
moddecl	::=	**Module** modbind
modbind	::=	*amodid* [**(**[paramlist] **)**][modop modexpr] [$\equiv$ modexpr]
modexpr	::=	bodyexpr [modop modexpr]
modop	::=	**:** | $\leq$ | $\geq$
bodyexpr	::=	overloadfun**(**modexpr **,** ...**,** modexpr **)**|
		pathid[**(**[iparamlist] **)**] |
		begin decl **end** |
		nil
paramlist	::=	params ;;; [**;** sharing]
params	::=	*amodid* [modop modexpr] | params **;** params
iparamlist	::=	modexpr | iparamlist **;** iparamlist
overloadfun	::=	**union**
		;;; | **intersection** | **complement** | **extension**
decl	::=	[hierarchy]
		[interface]
		[deductive]
		[control]
hierarchy	::=	**Open** modexpr |
		;;; **Sharing** patheq |
		Inherit pathid |
		moddecl |
		hierarchy hierarchy
interface	::=	[import]
		[export]
;;;sharing	::=	**sharing** patheq
pathid	::=	*amodid* | *amodid***/**pathid
patheq	::=	pathid $\equiv$ patheq | pathid $\equiv$ pathid |
		patheq **;** patheq
import	::=	**Import** predicateidlist
export	::=	**Export** predicateidlist
predicateidlist	::=	*predid* **,** predicateidlist | *predid*
pathpredid	::=	*predid* | *amodid***/**pathpredid

1.3 Deductive Knowledge

deductive	::=	**Deductive knowledge**
		[**Dictionary:** dictionary]
		[**Rules:** rules]
		[**Inference system:** logcomp]
		end deductive

1.3.1 Dictionary

dictionary	::=	[typedef] preddef
typedef	::=	**Types:** typebindings
typebindings	::=	*typeid* [**=** [**temporal**] typespec] |
		typebindings typebindings
typespec	::=	**(**valuesspec **)**|
		predefined |
		typeid |
		Array **[** arraydim **]** |
		Set Of *typeid* |
		frame framespec **end frame**
valuesspec	::=	*symbol* [string] [**(**number**,** number**,** number**,** number **)**] **,** valuesspec |
		symbol [string] [**(**number**,** number**,** number**,** number **)**]
predefined	::=	**boolean** | **logic** | **numeric** | **class**
arraydim	::=	*typeid* |
		numeric |
		arraydim **,** arraydim
framespec	::=	*slotid* **:** *typeid*
		framespec framespec
preddef	::=	**Predicates:** predbindings
predbindings	::=	*predid* **=** attributes |
		predbindings predbindings
attributes	::=	name [question] type [function] [relations] [explanation] [image]
name	::=	**Name:** *string*
question	::=	**Question:** *string*
type	::=	**Type:** [**temporal**] typespec
function	::=	**Function:** *S-expression*
relations	::=	**Relation:** *relationid pathpredid* |
		relations relations
relationid	::=	predef-rel-id | *symbol*
predef-rel-id	::=	**Needs** | **Needs_true** |
		Needs_false | **Belongs_to**

explanation	::=	**Explanation:** *string*
image	::=	**Image:** *string*

1.3.2 Rules

rules	::=	rule rules | rule
rule	::=	ruleid **If** premisse-rule **Then** conclusion-rule [*documentation*]
premisse-rule	::=	condition-rule **and** premisse-rule | condition-rule
conclusion-rule	::=	**conclude** rconclusion **is** cert-value
condition-rule	::=	conditio |
		meta-condition |
		no (conditio)|
		no (meta-condition)
rconclusion	::=	form |
		elemental ≡ (values)|
		no (elemental)|
		no (form *op* form)|
		no (elemental ≡ (values))
form	::=	elemental | (form *op* form)
elemental	::=	*predid* | *varid*
meta-condition	::=	conditio **is** conditio |
		is (conditio, conditio)|
		conditio **between** (conditio, conditio)|
		between (conditio, conditio, conditio)
values	::=	*symbol* , values | *symbol*
values1	::=	*symbol* **between** [*ltermid* , *ltermid*] , values1 |
		symbol **between** [*ltermid* , *ltermid*]
conditio	::=	operator (expression, ..., expression)|
		pathform | cert-value |
		expression operator expression
pathform	::=	pathform-s | pathform-c
pathform-s	::=	elemental | *amodid*/pathform-s | *varid*/pathform-s
pathform-c	::=	(formula)| *amodid*/pathform-c | *varid*/pathform-c
formula	::=	(pathform *op* pathform)
expression	::=	operator-arit (expression, ..., expression)|
		(expression operator-arit expression)|
		numeric | elemental | pathpredid | pathvar |
		values | values1
S-expression	::=	atom | list | S-expression S-expression

list ::= (S-expression)| ()
operator ::= < | > | <= | >= | = | /= | **int**
operator-arit ::= + | - | * | : | *symbol*

1.4 Inference System

logcomp ::= [lingtermdef] [order]
[renaming]
[connectives]
[infpat]
lingtermdef ::= **Truth values** = (ltermidlist)|
Truth values = (ltermlist)
ltermidlist ::= *ltermid* , ltermidlist | *ltermid*
ltermlist ::= *ltermid* = (real, real, real, real), ltermlist |
ltermid = (real, real, real, real)
cert-value ::= *ltermid* |
number |
[*ltermid* , *ltermid*] |
[*number* , *number*] |
(certarrows)
certarrows ::= certlist |
(certarrows)certarrows
certlist ::= *ltermid* |
number |
certlist certlist
renaming ::= **Renaming** lrenames
lrenames ::= pathid/cert-value ==> cert-value lrenames |
pathid/cert-value ==>cert-value |
S-expression
connectives ::= **Connectives:**
[**Negation** = fundef]
[**Conjunction** = fundef]
[**Disjunction** = fundef]
infpat ::= **Inference patterns:**
Modus ponens = fundef
fundef ::= S-expression | fileid | **luckasiewicz** |
Zadeh | **probabilistic** | truth-table
order ::= (ltermid, ltermid) order |
λ
truth-table ::= **Truth table** (arrows)

arrows	::=	<u>(</u>termlist<u>)</u> | arrows arrows
ltermlist	::=	ltermid | ltermlist ltermlist

1.5 Control Knowledge

control	::=	<u>**Control knowledge**</u> [search] [threshold][deduccnt] [structcnt] <u>**end control**</u>

1.5.1 Evaluation Type

search	::=	<u>**Evaluation type:**</u> evaltype
evaltype	::=	<u>**lazy**</u> | <u>**eager**</u> | <u>**heuristic**</u> | <u>**input**</u> | <u>**reified**</u>

1.5.2 Truth Threshold

threshold	::=	<u>**Truth threshold:**</u> cert-value

1.5.3 Deductive Control

deduccnt	::=	<u>**Deductive control:**</u> lmrr
lmrr	::=	mrr lmrr | mrr
mrr	::=	metaid <u>**If**</u> premisse-meta <u>**Then**</u> filters-mrr
filters-mrr	::=	filter-mrr filters-mrr | filter-mrr
premisse-meta	::=	condition-meta <u>**and**</u> premisse-meta | condition-meta
condition-meta	::=	mconditio | <u>**no**(</u>mconditio <u>)</u>|
mconditio	::=	metapredid <u>(</u>conditionterm <u>,</u> ..., conditionterm <u>)</u>
conditionterm	::=	operation <u>(</u>conditionterm <u>,...,</u> conditionterm <u>)</u>| metafunctid <u>(</u>conditionterm <u>,</u> ..., conditionterm <u>)</u>| conditio
filter-mrr	::=	<u>**inhibit rules**</u> relation-id pathpredid | <u>**inhibit rules**</u> pathpredid | <u>**prune**</u> pathpredid | <u>**increase**</u> form integer | <u>**decrease**</u> form integer | <u>**conclude**</u> conclusion-meta
conclusion-meta	::=	mconclusion | <u>**no**</u> <u>(</u>mconclusion <u>)</u>

mconclusion	::=	metapredid (conclusionterm , ..., conclusionterm)
conclusionterm	::=	operation (conclusionterm , ..., conclusionterm)|
		metafunctid (conclusionterm ,..., conclusionterm)|
		form

1.5.4 Structural Control

structcnt	::=	**Structural control:** lmre
lmre	::=	mre | mrx | lmre lmre
mre	::=	metaid **If** premisse-meta **Then** filter-mre
filter-mre	::=	**filter** amodidlist |
		order amodidlist **with certainty** cert-value |
		Open (conclusionterm , ..., conclusionterm)|
		Module (conclusionterm , ..., conclusionterm)|
		Inherit (conclusionterm , ..., conclusionterm)
amodidlist	::=	amodid amodidlist | amodid
mrx	::=	metaid **If** premisse-meta **Then** exception
exception	::=	**definitive solution** predid |
		stop