draft of reworking of 1989 documentation

An Overview of $\mathcal{NLL}$

Joachim Laubsch (hplabs), John Nerbonne (DFKI)

November 14, 1991

1 Language Constructs

1.1 Terms

Terms denote objects of the domain. Usually, a term picks a particular individual or set of the domain, but sometimes — as in the case of a *restricted parameter* — the choice is not determined.

1.1.1 Variables

⟨Variable⟩ ::= ?⟨Identifier⟩
denotes a variable, a totally undetermined object of the domain. Any occurrence of a variable may be captured, free or restrained (a terminology iftroduced by Barwise [?]). We will see language constructs that introduce such variables and possibly define the scope in which an occurrence is considered as captured, free or restrained.

1.1.2 Restricted Parameters

⟨Restricted Parameter⟩ ::= (⟨Variable⟩ | ⟨Restricting Wff⟩)
A restricted parameter[1] restrains a variable to an object of the domain for which the restricting formula holds. Any occurrence of the variable within some $\mathcal{NLL}$ expression that would otherwise be considered free as a consequence of introducing the restricted parameter — becomes a restrained occurrence. There may be more than one restricted parameter restraining the same variable.

1.1.3 Constants

⟨Constant⟩ ::= ⟨Identifier⟩ | ⟨String⟩
A constant denotes a totally determined object of the domain. Strings are employed in order to

[1]Notation: We use BNF syntax with non-terminals enclosed in "⟨⟩" brackets. Optional constituents are enclosed in "[]" brackets. Alternation is indicated by "|". This way we are free to use "|, [,]" as terminals of $\mathcal{NLL}$.

1. free the naming conventions to use arbitrary expressions, and
2. refer to names and formulate information about names

1.1.4 Function Terms

⟨Function Term⟩ ::= ⟨Function Name⟩(⟨Term⟩, ..)
⟨Function Name⟩ ::= ⟨Identifier⟩

> Example: `father(Jones)` denotes the unique individual who is the father of Jones.

A function name uniquely denotes a function. Functions have a fixed number of arguments. The position of a ⟨term⟩ in the list of supplied arguments identifies the argument to the function. Functions are typed and there is a way of associating a type signature with a function. For example the primitive functions

`in, on, near, at`

all have the signature:

> Physical_Object → Spatial_Region.
>
> Example: `in(California)` denotes the spatial region of the state of California.

For some binary functions, such as "`plus`" and "`times`" we introduce n-ary abbreviations, e.g. when $n > 2$

$$\texttt{plus}(x_1, \ldots x_n) \equiv \texttt{plus}(x_1, \texttt{plus}(x_2, \ldots x_n))$$

1.1.5 Location Terms

A special kind of function term is the location term. Such a term can either be formed by primitive functions (such as `in, on ...`) with range Spatial_Region or by intersecting spatial regions (see [?]).

> ⟨Location Term⟩ ::= reg-X{⟨Location Term⟩,⟨Location Term⟩}

reg-X is defined as the intersection operation over spatial regions. reg-X is associative and commutative:

> reg-X{reg-X{a,b},c} ≡ reg-X{a,reg-X{b,c}}
>
> reg-X{a,b} ≡ reg-X{b,a}}

We also use the abbreviated notation for an intersection of $n (n > 2)$ regions: reg-X{r_1, r_2, ... r_n} ≡ reg-X{r_1, reg-X{r_2 ... r_n} }

> Example: `work(agent: Jones loc: reg-X{in(Cal),at(IBM)})`

An Overview of $\mathcal{NLL}$

Joachim Laubsch (hplabs), John Nerbonne (DFKI)

November 14, 1991

1 Language Constructs

1.1 Terms

Terms denote objects of the domain. Usually, a term picks a particular individual or set of the domain, but sometimes — as in the case of a *restricted parameter* — the choice is not determined.

1.1.1 Variables

⟨Variable⟩ ::= ?⟨Identifier⟩
denotes a variable, a totally undetermined object of the domain. Any occurrence of a variable may be captured, free or restrained (a terminology iftroduced by Barwise [?]). We will see language constructs that introduce such variables and possibly define the scope in which an occurrence is considered as captured, free or restrained.

1.1.2 Restricted Parameters

⟨Restricted Parameter⟩ ::= (⟨Variable⟩ | ⟨Restricting Wff⟩)
A restricted parameter[1] restrains a variable to an object of the domain for which the restricting formula holds. Any occurrence of the variable within some $\mathcal{NLL}$ expression that would otherwise be considered free — as a consequence of introducing the restricted parameter — becomes a restrained occurrence. There may be more than one restricted parameter restraining the same variable.

1.1.3 Constants

⟨Constant⟩ ::= ⟨Identifier⟩ | ⟨String⟩
A constant denotes a totally determined object of the domain. Strings are employed in order to

[1]Notation: We use BNF syntax with non-terminals enclosed in "⟨⟩" brackets. Optional constituents are enclosed in "[]" brackets. Alternation is indicated by "|". This way we are free to use "|, [,]" as terminals of $\mathcal{NLL}$.

1. free the naming conventions to use arbitrary expressions, and
2. refer to names and formulate information about names

1.1.4 Function Terms

⟨Function Term⟩ ::= ⟨Function Name⟩(⟨Term⟩, ..)
⟨Function Name⟩ ::= ⟨Identifier⟩

> Example: `father(Jones)` denotes the unique individual who is the father of Jones.

A function name uniquely denotes a function. Functions have a fixed number of arguments. The position of a ⟨term⟩ in the list of supplied arguments identifies the argument to the function. Functions are typed and there is a way of associating a type signature with a function. For example the primitive functions

> `in, on, near, at`

all have the signature:

> Physical_Object → Spatial_Region.
>
> Example: `in(California)` denotes the spatial region of the state of California.

For some binary functions, such as "`plus`" and "`times`" we introduce n-ary abbreviations, e.g. when $n > 2$

$$\texttt{plus}(x_1, \ldots x_n) \equiv \texttt{plus}(x_1, \texttt{plus}(x_2, \ldots x_n))$$

1.1.5 Location Terms

A special kind of function term is the location term. Such a term can either be formed by primitive functions (such as `in, on ...`) with range Spatial_Region or by intersecting spatial regions (see [?]).

> ⟨Location Term⟩ ::= reg-X{⟨Location Term⟩,⟨Location Term⟩}

reg-X is defined as the intersection operation over spatial regions. reg-X is associative and commutative:

> reg-X{reg-X{a,b},c} ≡ reg-X{a,reg-X{b,c}}
>
> reg-X{a,b} ≡ reg-X{b,a}}

We also use the abbreviated notation for an intersection of $n (n > 2)$ regions: reg-X{r_1, r_2, ... r_n} ≡ reg-X{r_1, reg-X{r_2 ... r_n} }

> Example: `work(agent: Jones loc: reg-X{in(Cal),at(IBM)})`

An Overview of $\mathcal{NLL}$

Joachim Laubsch (hplabs), John Nerbonne (DFKI)

May 15, 1991

1 Introduction

$\mathcal{NLL}$ is a logical language for representing the meaning of natural language expressions. Its design has two goals: (1) to facilitate construction of the semantic structures from the syntactic and lexical structures, and (2) provide a formal base for disambiguation (e.g. DRT), and domain-specific interpretation (e.g. transduction into a database query language, such as SQL, or a language for controlling some application, such as the NewWave task language).

While the advantages of using logic for representing natural language meanings are widely appreciated, no consensus has been reached on the set of desirable features to be included in the logical language. The core of $\mathcal{NLL}$ — as presented in this report — constitutes common practice in computational linguistics (see [1]). We can see various extensions to this core in order to experiment with more controversial semantic issues, e.g. tense and temporal expressions, events and situations, or propositional attitudes.

We do not view $\mathcal{NLL}$ as "yet another representation language" but as a step towards a standard logical form usable by different language understanding systems. $\mathcal{NLL}$ is richer in expressive power than the currently developed "Knowledge Interchange Format" KIF ([7]), because $\mathcal{NLL}$ was designed to ease producing semantic forms. We can imagine a simple transduction phase that would map such $\mathcal{NLL}$ forms into KIF forms.

The next section describes the syntax and semantics of $\mathcal{NLL}$ constructs. The grammar of $\mathcal{NLL}$ is implemented using a YACC style parser/generator ([2]), and is summarized in section 3. Expressions of $\mathcal{NLL}$ can be constructed programmatically using the abstract syntax described in section 4.

2 Language Constructs

2.1 Terms

Terms denote objects of the domain. Usually, a term picks a particular individual or set of the domain, but sometimes — as in the case of a *restricted parameter* — the choice is not (fully) determined.

2.1.1 Constants

⟨Constant⟩ ::= ⟨Identifier⟩ | ⟨String⟩ | ⟨Numeral⟩

A constant denotes a totally determined object of the domain. Identifiers are used as tokens for objects of the domain. Strings are employed in order to

- free the naming conventions to use arbitrary expressions, and
- refer to names and formulate information about names

Syntactically, a string is a sequence of characters (except ") delimited by ".

Numerals denote the respective numbers. We rely here on the notation of the Common Lisp host language which provides integers, floating point numbers, rationals, and complex numbers.

2.1.2 Variables

⟨Variable⟩ ::= ?⟨Identifier⟩

denotes a variable, a totally undetermined object of the domain. Any occurrence of a variable may be captured, free or restrained (a terminology introduced by Barwise [3]). We will see language constructs that introduce such variables and possibly define the scope in which an occurrence is considered as captured, free or restrained.

2.1.3 Restricted Parameters

⟨Restricted Parameter⟩ ::= ([⟨Determiner⟩] ?⟨Identifier⟩ | ⟨Restriction⟩)
⟨Restriction⟩ ::= ⟨Wff⟩

A restricted parameter[1] restrains a variable to an object of the domain for which the restricting formula holds. Any occurrence of the variable within some $\mathcal{NLL}$ expression that would otherwise be considered free — as a consequence of introducing the restricted parameter — becomes a restrained occurrence. There may be more than one restricted parameter restraining the same variable. The optional determiner of a restricted parameter specifies the quantificational force of the restraint on the variable. The default quantificational force is the existential-determiner, but any determiner — like in a quantified Wff — can be used.

Example: "Jones hired two secretaries."
`hire(agent:Jones patient:(({= 2}) ?w | Secretary(instance:?w)))`

[1]Notation: We use BNF syntax with non-terminals enclosed in "⟨⟩" brackets. Optional constituents are enclosed in "[]" brackets. Alternation is indicated by "|". This way we are free to use "|, [,]" as terminals of $\mathcal{NLL}$.

The interpretation of a restricted parameter depends on a theory of discourse representation, which will assign in general several interpretations for formulas containing such variables. The interpretation of the above formula in terms of generalized quantifiers (see 2.3.3) yields the following $\mathcal{NLL}$ formula:

```
(({= 2}) ?w Secretary(instance:?w) hire(agent:Jones patient:?w))
```

The determiner "that" may be used as the quantificational force, as in:

"Jones hired that secretary."
`hire(agent:Jones patient:(that ?w | Secretary(instance:?w)))`

It is the task of a pragmatics module to find the object satisfying the restriction of that restricted parameter.

2.1.4 Function Terms

⟨Function Term⟩ ::= ⟨Function Name⟩(⟨Term⟩, ..)
⟨Function Name⟩ ::= ⟨Identifier⟩

> Example: `father(Jones)` denotes the unique individual who is the father of Jones.

A function name uniquely denotes a function. Functions have a fixed number of arguments. The position of a ⟨term⟩ in the list of supplied arguments identifies the argument to the function. Functions are typed and there is a way of associating a type signature with a function. For example the primitive functions

`in, on, near, at`

all have the signature: Physical_Object $\longrightarrow$ Spatial_Region.

> Example: `in(California)` denotes the spatial region of the state of California.

For some binary functions, such as "`plus`" and "`times`" we introduce n-ary abbreviations, e.g. when $n > 2$

$$\texttt{plus}(x_1, \ldots x_n) \equiv \texttt{plus}(x_1, \texttt{plus}(x_2, \ldots x_n))$$

2.1.5 Location Terms

A special kind of function term is the location term. Such a term can either be formed by primitive functions (such as `in, on` ...) with range Spatial_Region or by intersecting spatial regions (see [5]).

⟨Location Term⟩ ::= **reg-X**{⟨Location Term⟩,⟨Location Term⟩}
| ⟨Locative Function Term⟩
| ⟨Constant⟩
⟨Locative Function Term⟩ ::= ⟨Locative Function⟩(⟨Term⟩)
⟨Locative Function⟩ ::= in | on | at | near | ...

reg-X is defined as the intersection function over spatial regions,

reg-X: Spatial_Region × Spatial_Region → Spatial_Region

reg-X is associative and commutative.

reg-X{**reg-X**{a,b},c} ≡ **reg-X**{a,**reg-X**{b,c}}
reg-X{a,b} ≡ **reg-X**{b,a}

Since **reg-X** is associative we also use the abbreviated notation for an intersection of $n, (n > 2)$ regions:
reg-X{r_1, r_2, ... r_n} ≡ **reg-X**{r_1, **reg-X**{r_2 ... r_n} }

Example: `work(agent:Jones loc:reg-X{in(SanJose),in(Cal),at(IBM)})`

2.1.6 Measure Terms

⟨Measure Term⟩ ::= ⟨Specified Measure⟩
| ⟨Unspecified Measure⟩
| ⟨Maximally Specified Measure⟩
⟨Specified Measure⟩ ::= { ⟨Specifier⟩ ⟨Unspecified Measure⟩ }
⟨Unspecified Measure⟩ ::= ⟨Variable⟩ | ⟨Simple Measure⟩ | ⟨Complex Measure⟩

1. Simple Measures are denoted by numerals.
   ⟨Simple Measure⟩ ::= ⟨Numeral⟩

2. Complex Measures are composed of a magnitude (degree) and a scale.
   ⟨Complex Measure⟩ ::= [⟨Degree⟩ ⟨Scale⟩]
   ⟨Degree⟩ ::= ⟨Variable⟩ | ⟨Simple Measure⟩ | ⟨Specified Measure⟩
   ⟨Scale⟩ ::= 1 | foot | kg | ...

   Associated with a scale is a measure function μ (ranging over the natural or real numbers). We assume that μ is additive and multiplicative (see [9]).

   Example:

   "Jones is 6 feet tall."
   `tall(theme:Jones, measure:[6 foot])`

3. Specified Measures allow us to express *measure determiners* (see [9]).

   ⟨Specified Measure⟩ ::= { ⟨Specifier⟩ ⟨Unspecified Measure⟩ }
   ⟨Specifier⟩ ::= $< \mid \leq \mid = \mid \neq \mid > \mid \geq$

   Semantically, a Specified Measure is equivalent to a Restricted Parameter, as the following examples show:

   $$\{\leq [4\,l]\} \equiv (?m|?m \leq [4\,l])$$

   $$\{\leq 4\} \equiv (?m|?m \leq 4)$$

   But specified measures do not introduce parameters to which further reference is possible, and they are more compact in expression (cf. [9] for discussion).

   Example:

   "Jones is at most 6 feet tall."
   `tall(theme:Jones, measure:{ ≤ [6 foot]})`

4. Maximally Specified Measures differ from Specified Measures only in that they allow specifying a delta and a factor (e.g. the delta "2 more than" or the factor "twice as many as").

   ⟨Maximally Specified Measure⟩ ::=
   { ⟨Specifier⟩ ⟨Unspecified Measure⟩
   [**delta** {:⟨Unspecified Measure⟩ | :⟨Specified Measure⟩ }]
   [* :{ ⟨Simple Measure⟩ | ⟨Variable⟩ }] }

   Note that the argument for **delta** or * cannot be a Maximally Specified Measure (no recursion).

   Examples:

"more than 2 inches more than 6 feet"	`{> [6 foot] delta:[2 inch]}`
"twice as many as 3 kg"	`{= [3 kg] *:2}`
"a third as much as 53 mpg"	`{= [53 MPG] *:1/3}`

2.1.7 Plural Terms

Group Term ⟨Group Term⟩ ::= +{⟨Term⟩, ⟨Term⟩}

As the location constructor **reg-X** (see 2.1.5), the group constructor + is associative and commutative. We also use the abbreviated notation for the composition of $n (n > 2)$ groups:

$+\{g_1, g_2, \cdots g_n\} \equiv +\{g_1, +\{g_2 \cdots g_n\}\}$

Example: $+\{T, D, H\} \equiv +\{T, +\{D, H\}\}$

Sigma Term (sigma ⟨Variable⟩ | ⟨Restricting Wff⟩)

A sigma term designates a group of individuals — the sum of all individuals for which the restricting Wff holds. Sigma is a variable binding operator which restrains its variable like a restricted parameter does. A sigma term denotes a particular *individual sum* defined (by Link [8]) as

$$\sigma x P(x) = \iota x(^{*}P(x) \wedge \forall_y(^{*}P(y) \rightarrow y \leq x))$$

where * is an operator on 1-place predicates P which generates all the individual sums of members of the extension of P. Then $\sigma x P(x)$ denotes the supremum of $^{*}P(x)$.

Example: "All the men meet."
meet(theme/g:(sigma ?x | man(inst/i:?x)))

which expresses that the meet relation holds for the group consisting of all individuals which are instances of man. (The role "theme/g" is a specialisation of the "theme" role. The "/g" suffix indicates that its argument must be a plural term, see 2.3.1.)

A sigma term may be interpreted in terms of a restricted parameter, e.g. for the above example we have:

```
(?g | and{group(arg1:?g) man(inst/i:?g)
        (forall ?g1 man(inst/i:?g1)
                subgroup(arg1:?g1 arg2:?g))})
```

In general, in order to translate a sigma term, we create a restricted parameter for some group and require that this group is the maximal group.

If we want to express that the group has a particular size, we must use a restricted parameter instead of a sigma term:

Example:

```
"Jones hired the two secretaries."
hire(agent:Jones
     patient/i:(the ?g | and{group(arg1:?g)
                              Size(arg1:?g arg2:2)
                              Secretary(inst/i:?g)
                              Salient(arg1:?g)}))
```

This interpretation picks a salient group of two secretaries. Under a different interpretation, we would want to disallow any other group of two or more secretaries:

```
(the ?g | and{group(arg1:?g)
              Size(arg1:?g arg2:2)
              Secretary(inst/i:?g)
              (forall ?g1 Secretary(inst/i:?g1)
                      subgroup(arg1:?g1 arg2:?g))})
```

2.1.8 Complex Terms

In order to express aggregation operations on sets, s.a. finding the maximum or sum, we introduce terms which bind variables. The syntax of complex terms is:

```
⟨Complex Term⟩ ::= (⟨VB-TF-Operation⟩ ⟨Variable⟩, ...
                                      ⟨Operand⟩ | ⟨Wff⟩)
⟨VB-TF-Operation⟩ ::= *sum |*product |*min |*max |*avg
⟨Operand⟩ ::= ⟨Term⟩
```

A variable binding term former — ⟨VB-TF-Operation⟩ — is used to construct aggregations over the domain.

Example:

(*sum	?z, ?y	?y	|	sales(agt:?z jan$:?y))
operation	bound-vars	operand		restriction

Assuming that the 'sales' relation holds of agent ?z and jan$?y iff salesman ?z sells products worth ?y dollars in January, then the complex term refers to the total dollar sales for all salesmen in January.

Suppose we would model the relation 'sales' as a multi-set of tuples, called "Sales", and jan$ is the accessor to the **jan$** field of a tuple (similarly for **agent**). Then this is equivalent to

$$\sum_{s \in \text{Sales}} \text{jan\$}(s) \mid \exists z \; \text{agent}(s) = z$$

Note that it is important here that "Sales" denotes a multi-set, since there may be more than one january sale for a given salesman.

The operand part of a ⟨Complex Term⟩ is not limited to a single variable; instead, any ⟨Term⟩ can be used here as the following example shows:

(*sum ?z,?m1,?m2,?m3 plus(?m1,?m2,?m3) |
    sales(agt:?z jan$:?m1 feb$:?m2 mar$:?m3))

with the meaning

$$\sum_{s \in \text{Sales}} \text{jan\$}(s) + \text{feb\$}(s) + \text{mar\$}(s) \mid \exists z \; \text{agent}(s) = z$$

2.2 Predicates

2.2.1 Simple Predicates

Simple Predicates *may* be anadic (may have arbitrary arity, unlike functions). For each simple predicate a set of allowed arguments is specified. (The order of arguments is immaterial). The arguments are represented as roles (see 2.3.1). We allow an abbreviated notation which — in case no role names are given — supplies the default role names arg1, arg2, ... in the order the arguments are supplied.

Relational algebra (cf. [6], p. 265) introduces PROJECTIONS as a means of referring to relations of variable arity. A notational convenience for the use of anadic relations is that arguments to relations be identified via KEYWORDS rather than positions in argument vectors.

An example of an anadic predicate is the predicate "sale", defined to have the role set:

{agent, recipient,product,date,$val}

In $\mathcal{NLL}$ all of the following are well-formed:

```
sale(agt:obrien rcpnt:TRW product:887799 date:22Dec88 $val:18500)
sale(agt:obrien product:887799 date:22Dec88 $val:18500)
sale(rcpnt:TRW product:887799 date:22Dec88 $val:18500)
⋮
sale(rcpnt:TRW)
sale()
```

2.2.2 Predicate Constants

The following predicates are defined in the core of $\mathcal{NLL}$.

identity	`=(arg1:⟨Term⟩ arg2:⟨Term⟩)`
i-part	`i-part(theme:⟨Term⟩ in:⟨Plural Term⟩)`
size	`size(arg1:⟨Term⟩ arg2:⟨Plural Term⟩)`
atom	`atom(arg1:⟨Term⟩)`

2.2.3 Order Relation

Strict Order	`>(arg1:⟨Term⟩ arg2:⟨Term⟩)`
	`<(arg1:⟨Term⟩ arg2:⟨Term⟩)`

Partial Order `>=(theme:⟨Term⟩ pole:⟨Term⟩)`
`<=(theme:⟨Term⟩ pole:⟨Term⟩)`
`=(theme:⟨Term⟩ pole:⟨Term⟩)`

The order relations are defined on measure terms. An example of an order relation applied to a complex measure and a function term is:

```
>(theme:[6 feet] pole:plus([5 feet],[2 inch]))
```

2.2.4 λ-Predicates

⟨λ-Predicate⟩ ::= (lambda arg_1:⟨Variable⟩, ...,arg_n:⟨Variable⟩ ⟨Wff⟩)

λ-Predicates allow constructing a predicate from a Wff by abstracting over some free variables of the Wff. This is desirable in disambiguation where a predicate variable may become bound later by some λ-Predicate.

Examples:

- VP ellipsis

  ```
  "Jones works at IBM and Smith does too."
  ṗ =def (λ arg₁:?x work(agent:?x loc:at(IBM)))
  and{ṗ(arg₁:Jones) ṗ(arg₁:Smith) }
  ```

  Here $\dot{p}$ is a meta variable, not an entity of $\mathcal{NLL}$. It belongs to the meta-language[2]. That is, all the occurrences of $\dot{p}$ in the final line must be replaced by the expression, which is the definition of $\dot{p}$ to produce the $\mathcal{NLL}$ form for the sentence.

- Predicative Traces

  ```
  "How tall did Smith think that Jones is _ ?"
  (?lambda ?w
    (lambda arg1:?P
            think(agt:Smith theme:↑(?P(Jones))))
    (arg1:↑(lambda(arg1:?x) tall(theme:?x spec:?w))))
  ```

- Type raising

  ```
  "Smith and every manager attended."
  Ṗ =def (lambda arg₁:?x attended(theme:?x))
  and{Ṗ(arg1:Smith)
      (forall ?m Manager(inst:?m) Ṗ(arg1:?m))}
  ```

[2]During interim stages of semantics processing we use such variables, simply by adopting some naming convention.

- $\overline{N}$ Anaphora

```
   "The tall student talked and the short one listened."
The analysis comes in two parts
1  (the ?x and{Student(inst:?x) tall(theme:?x)}
          talk(agent:?x))
2  (the ?y and{Ṗ(arg1:?y) short(theme:?y)}
          listen(agent:?y))
```

We now find a plausible definition for $\dot{P}$:

$\dot{P} =_{def}$ `(lambda` arg_1`:?x student(inst:?x))`

Compare this to:

```
   "Several tall students talked.  One listened."
1  (Several ?x and{Student(inst:?x) tall(theme:?x)}
              talk(agent:?x))
2  (({= 1}) ?y Ṗ(arg1:?y)
              listen(agent:?y))
```

We find as a plausible definition for $\dot{P}$ the abstraction of the restriction set

$\dot{P} =_{def}$ `(lambda` arg_1`:?x student(inst:?x) tall(theme:7x))`

2.2.5 Complex Predicates

⟨Complex Predicate⟩ ::= ⟨Predicate Operator⟩ ⟨Simple Predicate⟩
⟨Predicate Operator⟩ ::= -er | -as | -less | -est | -least

A predicate operator may be applied to predicates which are *gradable*. Such a predicate P has associated a measure function with range $\mathcal{M}_P$ ordered by $<_P$. The set of roles defined for P contains at least the roles "theme", "pole", and "spec".

Example:

```
"Jones is 2 inches taller than 6 feet."
-er(tall)(theme:Jones pole:[6 feet] spec:[2 inch]) ≡
tall(theme:Jones
     spec:(?h | >(theme:?h
                  pole:plus([6 feet],[2 inch]))))
```

The meaning of the predicate operator "-er" is:

$$\texttt{-er}(P)(\texttt{theme:}x\ \texttt{pole:}p\ \texttt{spec:}d) \equiv$$
$$\exists_{m \in \mathcal{M}_P}\ p <_P m \wedge |m - p| = d \wedge P(\texttt{theme:}x\ \texttt{spec:}m)$$

Similarly, the meaning of the predicate operator "-less" is:

$$\texttt{-less}(P)(\texttt{theme:}x\ \texttt{pole:}p\ \texttt{spec:}d) \equiv$$
$$\exists_{m \in \mathcal{M}_P}\ m <_P p \wedge |m - p| = d \wedge P(\texttt{theme:}x\ \texttt{spec:}m)$$

2.3 Well-formed Formulas

2.3.1 Atomic Wffs

⟨Atomic Wff⟩ ::= ⟨Predicate⟩(⟨Role-Argument Pair⟩ ...)
⟨Role-Argument Pair⟩ ::= ⟨Role⟩ : ⟨Term⟩
⟨Role⟩ ::= ⟨Simple Role⟩ | ⟨Complex Role⟩

Simple Roles The core of $\mathcal{NLL}$ defines many simple roles (see 2.3.1), but the set of simple roles is open. An application will define predicates and roles particular to a domain.

Since predicates are anadic, the question arises what the meaning of an atomic wff is which supplies less roles than are defined for the predicate. (If a role is supplied that is not defined for the predicate, or if a role is multiply supplied, the meaning of the atomic wff will be undefined.) For all the unsupplied roles values are assumed to exist:

Consider the $\mathcal{NLL}$ expression $P(r_1 : a_1 \ldots r_n : a_n)$ and let $R(P)$ be the roles defined for P, and $\{s_1 \ldots s_m\}$ be the set of roles in $R(P)$ but not supplied, i.e.

$$\{s | s \in R(P) \wedge \neg\exists_{i=1 \ldots n} r_i = s\}$$

then

$$P(r_1 : a_1 \ldots r_n : a_n) \equiv$$
$$\exists_{v_1 \ldots v_m} P(r_1 : a_1 \ldots r_n : a_n, s_1 : v_1 \ldots s_m : v_m)$$

Example: Suppose the "eat" predicate has the defined roles "eater", "eaten" — and no other roles. Then it follows from the semantics of $\mathcal{NLL}$ that:

```
eat(eater:John) ≡ ∃?x eat(eater:John eaten:?x)
```

Complex Roles Some roles can take as an argument either an individual or a group term (see 2.1.7). In order to refer to the individual members of the group, we use the suffix "/i", and in order to refer to the group we use the suffix "/g".

⟨Complex Role⟩ ::= ⟨Individual Role⟩ | ⟨Group Role⟩
⟨Individual Role⟩ ::= ⟨Simple Role⟩ /i
⟨Group Role⟩ ::= ⟨Simple Role⟩ /g
Examples:

```
"Tom, Dick and Harry carry a piano."
carry(agent/g:+{Tom, Dick, Harry} theme:(?x | Piano(inst:?x)))
```

The complex role **agent/g** indicates that Tom, Dick and Harry collectively carry a piano (the *collective* reading).

```
"Tom, Dick and Harry sing."
sing(agent/i:+{Tom, Dick, Harry})
```

Here, **agent/i** indicates that each of the individual members of the group sings (the *distributive* reading). A special rule would make this distributive inference explicit:

```
and{sing(agent:Tom) sing(agent:Dick) sing(agent:Harry)}
```

Some verbs allow only the collective reading. Such a verb is 'meet':

```
"The managers meet in room 'Chaos'."
meet(agent/g:(sigma ?m | Manager(inst:?m))
     loc:in((?l | Room(inst:?l name:'Chaos'))))
```

2.3.2 Non-atomic WFF

⟨Non-atomic WFF⟩ ::= ⟨Negation⟩ | ⟨Conditional-Wff⟩ | ⟨N-ary-Conn-Wff⟩
⟨Negation⟩ ::= ~ ⟨Wff⟩
⟨Conditional-Wff⟩ ::= if (⟨Wff⟩ ⟨Wff⟩ [⟨Wff⟩])
⟨N-ary-Conn-Wff⟩ ::= ⟨N-ary-Connective⟩ {⟨Wff⟩ ...}
⟨N-ary-Connective⟩ ::= and | or | iff | xor

The meaning of these is as usual in predicate logic.
Examples:

```
"Abrams does not hire Browne."
~hire(agent:Abrams patient:Browne)

"Jones either works or manages Smith."
xor{works(agt:Jones) manage(agt:Jones pat:Smith)}
```

```
"If Abrams hired Browne, he manages him."
if(hire(agent:Abrams patient:Browne)
   manage(agent:?a patient:?a1))
```

2.3.3 Quantified Wff

⟨Quantified Wff⟩ ::= (⟨Quantifier⟩ ⟨Scope⟩)
⟨Quantifier⟩ ::= ⟨Determiner⟩ ⟨Variable⟩ ... ⟨Restrictor⟩
⟨Quantifier⟩ ::= ⟨Ternary Quantifier⟩
⟨Ternary Quantifier⟩ ::= ⟨Order Relation⟩ ⟨Variable⟩ ...
    ⟨Restrictor⟩ ⟨Pole Restrictor⟩
⟨Restrictor⟩ ::= ⟨Wff⟩
⟨Pole Restrictor⟩ ::= ⟨Wff⟩
⟨Scope⟩ ::= ⟨Wff⟩
⟨Determiner⟩ ::= ⟨Simple Determiner⟩ | ⟨Complex Determiner⟩
⟨Simple Determiner⟩ ::= exists | forall | the | most | several | max | min
⟨Complex Determiner⟩ ::= (⟨Maximally Specified Measure⟩)

Generalized quantifiers: The notion of a generalized quantifier[3] is based on the relation between two sets – a restriction set R and a body set B. $R \cap B$ is called the intersection set I. Within $\mathcal{NLL}$, a generalized quantifier Q is defined as a binary predicate[4]:

$$\lambda(n, m)Q(n, m)$$

which is applied to the cardinality of the restriction set, $|R|$, and the cardinality of the intersection set $|R \cap B|$. This number-theoretic characterization of generalized quantifiers is adequate for finite sets. There is an equivalent formulation in terms of sets and relations among sets (for a comparison see [10]).

Syntactically, a Generalized Quantifier is expressed by ⟨Quantifier⟩ and determines the Restriction set (R). The ⟨Scope⟩ of a ⟨Quantified Wff⟩ determines the body set (B). The meaning of a Quantified Wff can be expressed as a relation involving R and the intersection set $I = R \cap B$[5].

Existential Determiner: `exists ?x` $\phi(?x)$

If the ⟨Scope⟩ of the ⟨Quantified Wff⟩ is given by $\psi(?x)$ and only ?x occurs free in $\phi(?x)$ and $\psi(?x)$, then

[3] We include only natural language quantifiers which are "quantificational", excluding possessives, vague quantifiers, and quantifiers expressing defaults (such as "usually").

[4] For a comprehensive introduction and survey of quantifiers in logic and linguistics see [10].

[5] In the following, we present the number-theoretic characterization of the quantifiers, with the set-theoretic ones in parentheses.

(**exists** ?x $\phi(?x)\psi(?x)$) is true iff
$|R \cap S| > 0$ (or $R \cap B \neq \emptyset$)
where $R = \{?x|\phi(?x)\}$ and $B = \{?x|\psi(?x)\}$.

Universal Determiner: forall ?x $\phi(?x)$

(**forall** ?x $\phi(?x)\psi(?x)$) is true iff $|R| = |I|$ (or $R \subseteq B$)

Most: most ?x $\phi(?x)$

(**most** ?x $\phi(?x)\psi(?x)$) is true iff $|I| > |R|/2$

We are unable to represent **most** in terms of set-theoretic relations: the given semantics of **most** assumes that both I and R are *finite* sets. The same applies to the following quantifiers.

Several: several ?x $\phi(?x)$

(**several** ?x $\phi(?x)\psi(?x)$) is true iff $|I| > 1$

The: the ?x $\phi(?x)$

(**the** ?x $\phi(?x)\psi(?x)$) is true iff $|R| = |I| = 1$

Max: max ?x $\phi(?x)$

(**max** ?x $\phi(?x)\psi(?x)$) is true iff $|R| > 0 \wedge \psi(max(R))$

where *max* is a function that selects the largest element from a set of numbers.

Min: min ?x $\phi(?x)$

(**min** ?x $\phi(?x)\psi(?x)$) is true iff $|R| > 0 \wedge \psi(min(R))$

where *min* is a function that selects the smallest element from a set of numbers.

Complex determiners (Maximally Specified Measures):

(($\{= 1\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| = 1$
(($\{= 2\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| = 2$
...
similarly for the specifiers $<, \leq, \neq, >, \geq$
(($\{< ?n\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| < ?n$
(($\{\leq ?n\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| \leq ?n$
(($\{\neq ?n\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| \neq ?n$
(($\{> ?n\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| > ?n$
(($\{\geq ?n\}$) ?x $\phi(?x)$ $\psi(?x)$) is true iff $|I| \geq ?n$

Example: "More than five men smoke."

```
(({> 5}) ?x man(inst:?x) smoke(agent:?x))
```

This quantified wff is true iff the intersection of the set of men and the set of objects that smoke is greater than 5.

Note that the Unspecified Measure part of the Specified Measure may be a variable. This is useful in

Comparatives: "More consultants work than engineers teach."

```
(exists ?n NatNum(inst:?n)
    and{({(= ?n)} ?m Engineer(inst:?m) teach(agt:?m))
        ({(> ?n)} ?c Consultant(inst:?c) work(agt:?c))})
```

"How many" questions: "How many men smoke?"[6]

```
(?lambda ?m Measure(inst:?m)
   (({= ?m}) ?x man(inst:?x) smoke(agent:?x)))
```

In the above cases, the determiner's Unspecified Measure has always been a Simple Measure, but it may be a Complex Measure (having a degree and a scale), as in the following example:

"At most 3 l oil spilled."

```
(({≤ [3 l]}) ?x Oil(inst:?x) Spill(theme:?x))
```

The Complex Measure [3 l] contains the scale l. We assume that there is a function Quantity$_l$ which returns the quantity in liters, associated with a mass object. Then the above is satisfied iff

$$\sum_{i \in I} Quantity_l(i) \leq 3$$

where I = { ?x | Oil(inst:?x)} ∩ {?x | Spill(theme:?x)}.

Similarly, the magnitude of the complex measure may be a variable:

```
"Between 3 and 5 l oil spilled."
  (exists ?n NatNum(inst:?n)
    and{(({= [?n l]}) ?x oil(inst:?x) spill(theme:?x))
        >=(theme:?n pole:3)
        <=(theme:?n pole:5)})
```

[6](see section 2.3.4)

Ternary Determiners $>$?x $\phi_1(?x)$ $\phi_2(?x)$

Let

$$\phi_1(?x) \cap \psi(?x) = I_1$$

$$\phi_2(?x) \cap \psi(?x) = I_2$$

Then ($>$?x $\phi_1(?x)$ $\phi_2(?x)$ $\psi(?x)$) is true iff $|I_1| > |I_2|$

Similarly, for the other order relations `>= < <=: =`
($>=$?x $\phi_1(?x)\,\phi_2(?x)\,\psi(?x)$) is true iff $|I_1| \geq |I_2| \ldots$

Example:

```
"More men than women smoke."
  (> ?x man(inst:?x) woman(inst:?x) smoke(agt:?x))
  ≡
  (max ?n smoke(agt:(({= ?n}) ?w | woman(inst:?w)))
     (exists ?m >(standard:?m pole:?n)
        smoke(agt:(({= ?m}) ?x | man(inst:?x)))))
```

In words: if `?n` were the maximal number of women who smoke, there is a larger number, `?m`, of men who smoke.

2.3.4 Question Wff

⟨Question Wff⟩ ::= (⟨Questioner⟩ ⟨Scope⟩)
⟨Questioner⟩ ::= ?lambda ⟨Variable⟩, ... ⟨Restriction⟩
⟨Scope⟩ ::= ⟨Wff⟩
⟨Restriction⟩ ::= ⟨Wff⟩

The meaning of (?lambda ?x $\phi(?x)$ $\psi(?x)$) is a predicate characterizing the set of all true propositions that arise from substituting `?x` by some denoting term n in

$$\text{and}\{\phi(?x)\psi(?x)\}$$

Identifying the meaning of a question with the set of all true possible answers gives us a set which is too large, and it must be further reduced on pragmatic grounds. One pragmatic rule would for instance state that the term n carries information — allows the reduction of the set of possible referents of n.[7]

Examples:

```
"What did O'Brian sell?"
(?lambda ?x Thing(inst:?x) sale(agent:O'Brian product:?x))
```

[7] An example of an uninformative answer to the question: "Who bought the book?" would be: "the one who bought the book".

```
"How many copies of Advancelink did IG-Farben buy in December?"
(?lambda ?n NatNum(inst:inst:?n)
         sale(product:Advancelink
              recipient:Igfarben
              date:December))

"Which woman manages what department?"
(?lambda ?x,?y
         and{woman(inst:?x) department(inst:?y)}
         manage(agt:?x pat:?y))
```

2.3.5 Examples

```
"A child walks."
(exists ?x child(inst:?x) walk(agent:?x))
"All children walk."
(forall ?x child(inst:?x) walk(agent:?x))
"Several children walk."
(several ?x child(inst:?x) walk(agent:?x))
"Most children walk."
(most ?x child(inst:?x) walk(agent:?x))
"The child walks."
(the ?x child(inst:?x) walk(agent:?x))
"One child walks."
((= 1) ?x child(inst:?x) walk(agent:?x))
"Three children walk."
({> 3} ?x child(inst:?x) walk(agent:?x))

"At least 3 but no more than 6 children sing."

 sing(agent:(sigma ?x |
                and{Child(inst/i:?x)
                    >=(standard:(?n | Size(arg1:?x arg2:?n)) pole:3)
                    ~>(standard:?n pole:6)}))

"Three liters of oil spilled."
({= [3 l]} ?x oil(inst:?x) spill(theme:?x))

"Exactly two more consultants work than MTSs teach."

(exists ?n NatNum(inst:?n)
   and{({= ?n} ?m Mts(inst:?m) teach(agent:?m))
       ({> ?n delta:2} ?c Consultant(inst:?c)
                          work(agent:?c))})
```

"At least two more consultants work than MTSs teach."

```
(exists ?n NatNum(inst:?n)
   and{({= ?n} ?m Mts(inst:?m) teach(agent:?m))
       ({> ?n delta:(?d | ?d ≥ 2)} ?c Consultant(inst:?c)
                                   work(agent:?c))})
```

"Exactly twice as many consultants work as MTSs teach."

```
(exists ?n NatNum(inst:?n)
   and{({= ?n} ?m Mts(inst:?m) teach(agent:?m))
       ({> ?n *:2} ?c Consultant(inst:?c)
                   work(agent:?c))})
```

"Exactly half as many consultants work as MTSs teach."

```
(exists ?n NatNum(inst:?n)
   and{({= ?n} ?m Mts(inst:?m) teach(agent:?m))
       ({> ?n *:1/2} ?c Consultant(inst:?c)
                     work(agent:?c))})
```

3 BNF of $\mathcal{NLL}$'s Concrete Syntax

```
⟨Wff⟩ ::= ⟨Atomic Wff⟩
        | ⟨Non-atomic WFF⟩
        | ⟨Quantified Wff⟩
        | ⟨Question Wff⟩


⟨Atomic Wff⟩ ::= ⟨Predicate⟩(⟨Role-Argument Pair⟩ ...)
               | ⟨Propositional Variable⟩
               | true | false
⟨Role-Argument Pair⟩ ::= ⟨Role⟩:⟨Term⟩
⟨Role⟩ ::= ⟨Simple Role⟩ | ⟨Complex Role⟩
⟨Simple Role⟩ ::= agent | theme | patient | apprehender | goal | beneficiary
                | for | to | from | within | with | location
                | date | time | direction | destination
                | at | in | on | by | under | beside | over
                | proposition | situation | result | event | state
                | instance | of | possessor | possessed | bearer
                | measure | specifier | pole | difference | standard
                | key | relatum1 | relatum2 | np-n-relatum
                | arg | arg1 | arg2 | arg3 | arg4 | arg5 | arg6
⟨Complex Role⟩ ::= ⟨Individual Role⟩ | ⟨Group Role⟩
⟨Individual Role⟩ ::= ⟨Simple Role⟩ /i
⟨Group Role⟩ ::= ⟨Simple Role⟩ /g
⟨Propositional Variable⟩ ::= ⟨Identifier⟩


⟨Non-atomic WFF⟩ ::= ⟨Negation⟩ | ⟨Conditional-Wff⟩ | ⟨N-ary-Conn-Wff⟩
⟨Negation⟩ ::= ~⟨Wff⟩
⟨Conditional-Wff⟩ ::= if ( ⟨Wff⟩ ⟨Wff⟩ [ ⟨Wff⟩ ] )
⟨N-ary-Conn-Wff⟩ ::= ⟨N-ary-Connective⟩ {⟨Wff⟩ ...}
⟨N-ary-Connective⟩ ::= and | or | iff | xor


⟨Term⟩ ::= ⟨Variable⟩
         | ⟨Constant⟩
         | ⟨Function Term⟩
         | ⟨Location Term⟩
         | ⟨Measure Term⟩
         | ⟨Plural Term⟩
         | ⟨Restricted Parameter⟩
```

```
           | ⟨Complex Term⟩

⟨Variable⟩ ::= ?⟨Identifier⟩
⟨Constant⟩ ::= ⟨Identifier⟩ | ⟨String⟩ | ⟨Numeral⟩
⟨Function Term⟩ ::= ⟨Function Name⟩(⟨Term⟩, ...)
⟨Function Name⟩ ::= ⟨Identifier⟩
⟨Location Term⟩ ::= reg-X{⟨Term⟩, ...}

⟨Measure Term⟩ ::=
      ⟨Unspecified Measure⟩ | ⟨Specified Measure⟩ | ⟨Maximally Specified Measure⟩
⟨Unspecified Measure⟩ ::=
      ⟨Variable⟩ | ⟨Simple Measure⟩ | ⟨Complex Measure⟩
⟨Simple Measure⟩ ::= ⟨Numeral⟩
⟨Complex Measure⟩ ::= [ ⟨Degree⟩ ⟨Scale⟩ ]
⟨Degree⟩ ::= ⟨Variable⟩ | ⟨Simple Measure⟩ | ⟨Specified Measure⟩
⟨Scale⟩ ::= l | foot | kg | ...
⟨Specified Measure⟩ ::= { ⟨Specifier⟩ ⟨Unspecified Measure⟩ }
⟨Specifier⟩ ::= ⟨Order Relation⟩
⟨Order Relation⟩ ::= < | ≤ | = | > | ≥
⟨Maximally Specified Measure⟩ ::=
      { ⟨Specifier⟩ ⟨Unspecified Measure⟩
        [ delta {:⟨Unspecified Measure⟩ | :⟨Specified Measure⟩ }]
        [ *:{ ⟨Simple Measure⟩ | ⟨Variable⟩ }] }

   ⟨Plural Term⟩ ::= ⟨Group Term⟩ | ⟨Sigma Term⟩

   ⟨Group Term⟩ ::= +{⟨Term⟩, ... }

   ⟨Sigma Term⟩ ::= (sigma ?⟨Identifier⟩ | ⟨Wff⟩)

   ⟨Restricted Parameter⟩ ::= ([⟨Determiner⟩] ?⟨Identifier⟩ | ⟨Restriction⟩)
⟨Restriction⟩ ::= ⟨Wff⟩

⟨Complex Term⟩ ::=
      (⟨VB-TF-Operation⟩ ⟨Variable⟩, ... ⟨Operand⟩ | ⟨Wff⟩)
⟨VB-TF-Operation⟩ ::= *sum | *product | *min | *max | *avg

⟨Predicate⟩ ::= ⟨Simple Predicate⟩ | ⟨Complex Predicate⟩ | ⟨λ-Predicate⟩
⟨Simple Predicate⟩ ::= ⟨Identifier⟩
```

⟨Complex Predicate⟩ ::= ⟨Predicate Operator⟩ ⟨Simple Predicate⟩
⟨Predicate Operator⟩ ::= -er | -as | -less | -est | -too | -least
⟨λ-Predicate⟩ ::= (λ arg_1:⟨Variable⟩, ... arg_n:⟨Variable⟩ ⟨Wff⟩)

⟨Quantified Wff⟩ ::= (⟨Quantifier⟩ ⟨Scope⟩)
⟨Quantifier⟩ ::= ⟨Determiner⟩ ⟨Variable⟩ ... ⟨Restrictor⟩
⟨Quantifier⟩ ::= ⟨Ternary Quantifier⟩
⟨Ternary Quantifier⟩ ::= ⟨Order Relation⟩ ⟨Variable⟩ ... ⟨Restrictor⟩ ⟨Pole Restrictor⟩
⟨Restrictor⟩ ::= ⟨Wff⟩
⟨Pole Restrictor⟩ ::= ⟨Wff⟩
⟨Scope⟩ ::= ⟨Wff⟩
⟨Determiner⟩ ::= ⟨Simple Determiner⟩ | ⟨Complex Determiner⟩
⟨Simple Determiner⟩ ::= exists | forall | the | most | several | max | min
⟨Complex Determiner⟩ ::= (⟨Maximally Specified Measure⟩)

⟨Question Wff⟩ ::= (⟨Questioner⟩ ⟨Scope⟩)
⟨Questioner⟩ ::= ?lambda ⟨Variable⟩, ... ⟨Restriction⟩
⟨Scope⟩ ::= ⟨Wff⟩
⟨Restriction⟩ ::= ⟨Wff⟩

4 $\mathcal{NLL}$'s Abstract Syntax

For each defined type of structure we list its slots (by convention a slot's name begins with "-") and the type of argument filling this slot. Since slots are inherited — to get all slots for a subtype, consult the type hierarchy in section 4.2 and form the union of all the slots from the subtype up to the root NLL-Domain. I.e. every type has the slot "-extras" which can be used by the implementation to store additional information. This slot should be of the type association-list. The non-terminals of the grammar usually have a type of the same name associated with them, and studying that grammar rule and the slots for the type should make it obvious how the semantic action for that BNF rule should be expressed (e.g. in YACC).

4.1 Structures and their Slots

4.1.1 NLL-Domain

[NLL-Domain [-Extras Association-List]]

4.1.2 NLL-Term

NLL-Const	-Name	{Symbol | String}
NLL-Integer	-Att	Integer
NLL-Real	-Att	Real
NLL-Var-Occ	-Name	Symbol
	-Bound-Var	NLL-Bound-Var
Function-Term	-Function	Simple-Predicate
	-Arguments	Seq(NLL-Term)
Location-Term	-Component-terms	NLL-Term
Group-Term	-Component-terms	NLL-Term

4.1.3 Complex-Term

Complex-Term	-Operation	VB-TF-Operation
	-Bound-Vars	NLL-Bound-Var
	-Operand	NLL-Term
	-Restriction	NLL-Wff

4.1.4 NLL-Bound-Var

NLL-Bound-Var	-Name	Symbol
	-References	Set(NLL-Var-Occ)
	-Scope	NLL-Expr

4.1.5 Restricted-Parameter

$$\left[\begin{array}{ll}\text{Restricted-Parameter} & \left[\begin{array}{ll}\text{-Var} & \text{NLL-Bound-Var}\\ \text{-Restriction} & \text{NLL-Wff}\\ \text{-Quantificational-Force} & \text{QForm-Determiner}\end{array}\right]\end{array}\right]$$

4.1.6 Measure

$$\left[\begin{array}{ll}\text{Specified-Measure} & \left[\begin{array}{ll}\text{-Specifier} & \text{Order-Relation}\\ \text{-Numeric} & \text{NLL-Term}\end{array}\right]\\ \text{Maximally-Specified-Measure} & \left[\begin{array}{ll}\text{-Specifier} & \text{Order-Relation}\\ \text{-Numeric} & \text{NLL-Term}\\ \text{-Delta} & \{\text{Specified-Measure} \mid \text{Complex-Measure}\}\\ \text{-Factor} & \{\text{NLL-Var-Occ} \mid \text{Unspecified-Numeric}\}\end{array}\right]\\ \text{Complex-Measure} & \left[\begin{array}{ll}\text{-Magnitude} & \text{NLL-Term}\\ \text{-Scale} & \text{Symbol}\end{array}\right]\end{array}\right]$$

4.1.7 Role

$$\left[\begin{array}{ll}\text{Role} & \left[\begin{array}{ll}\text{-Name} & \text{Symbol}\end{array}\right]\\ \text{Role-Complex} & \left[\begin{array}{ll}\text{-Role} & \text{Role}\end{array}\right]\\ \text{Role-Variable-Pair} & \left[\begin{array}{ll}\text{-Role} & \text{Role-Complex}\\ \text{-Variable} & \text{NLL-Bound-Var}\end{array}\right]\\ \text{Role-Argument-Pair} & \left[\begin{array}{ll}\text{-Role} & \text{Role-Complex}\\ \text{-Argument} & \text{NLL-Term}\end{array}\right]\end{array}\right]$$

4.1.8 Predicate

$$\left[\begin{array}{ll}\text{Simple-Predicate} & \left[\begin{array}{ll}\text{-Name} & \text{Symbol}\end{array}\right]\\ \text{Complex-Predicate} & \left[\begin{array}{ll}\text{-Predicate} & \text{Simple-Predicate}\\ \text{-Operator} & \text{Predicate-Operator}\end{array}\right]\\ \text{Lambda-Predicate} & \left[\begin{array}{ll}\text{-Lambda-List} & \text{Set(Role-Variable-Pair)}\\ \text{-Scope} & \text{NLL-Wff}\end{array}\right]\end{array}\right]$$

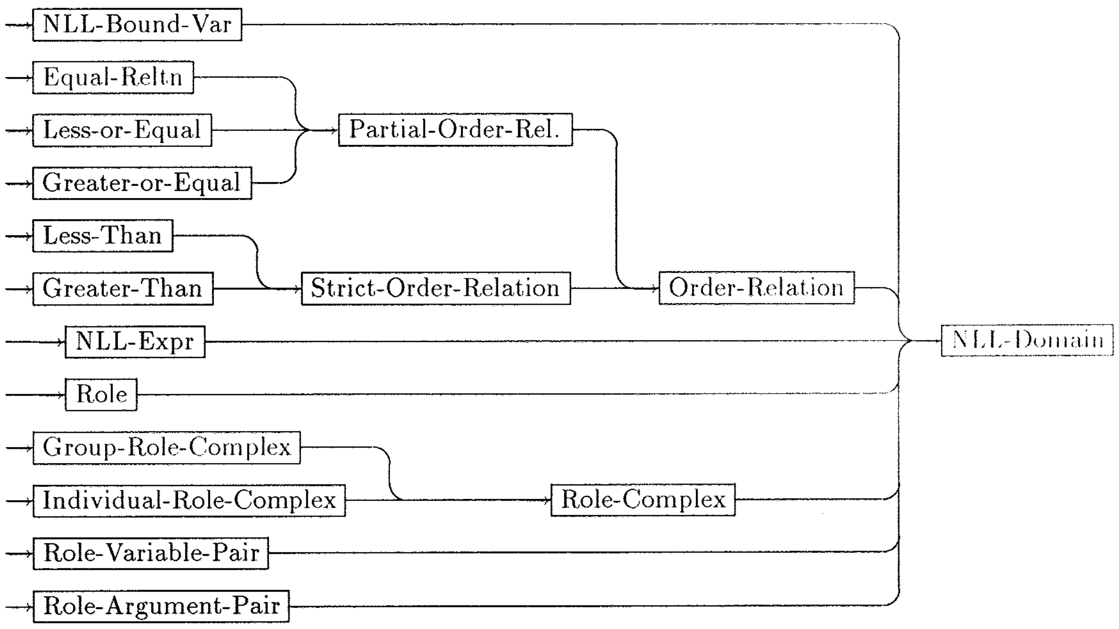
4.1.9 Well-formed Formula

Propositional-Variable	-Name	Symbol
Atomic-Wff	-Predicate	NLL-Predicate
	-Role-Argument-Pairs	Set(Role-Argument-Pair)
N-ary-Conn-Wff	-Connective	N-ary-Conn
	-Subformula-Set	Set(NLL-Wff)
Negated-Wff	-Scope	NLL-Wff
Conditional-Wff	-Connective	Conditional
	-Antecedent	NLL-Wff
	-Consequent	NLL-Wff
	-Else-Clause	NLL-Wff
Propositional-Sense	-Conceptual-Level	Integer
	-Proposition	NLL-Wff
Qform-Wff	-Qform	Qform
	-Scope	NLL-Wff
Qform	-Determiner	Determiner
	-Vars	Seq(NLL-Bound-Var)
	-Restrictor	NLL-Wff
Ternary-Quantifier	-Pole-restrictor	NLL-Wff
Simple-Determiner	-Name	Symbol
Complex-Determiner	-Specified-Measure	Specified-Measure

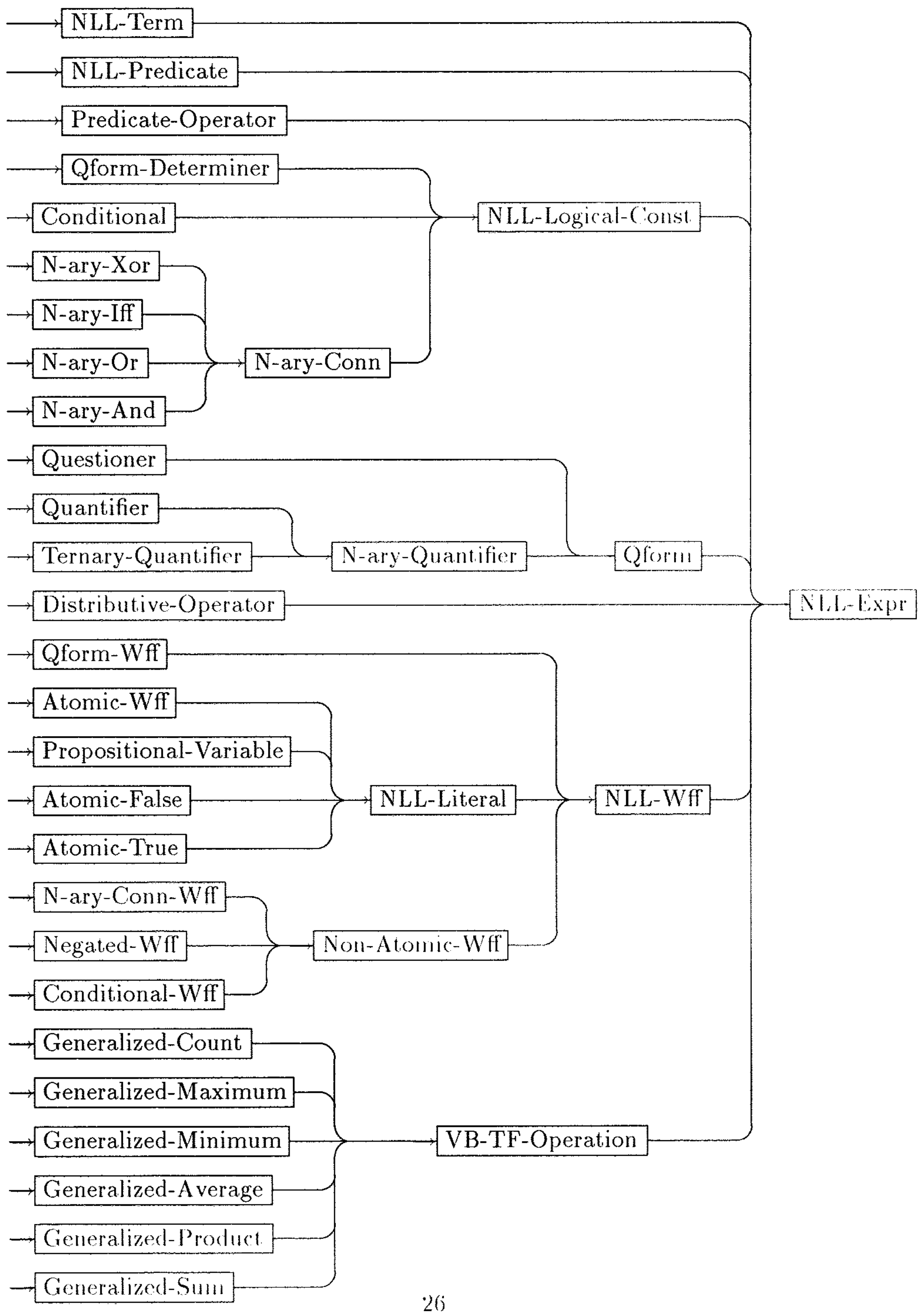
4.2 $\mathcal{NLL}$'s Type Hierarchy

The root of the type hierarchy is NLL-Domain. Leaf nodes of the hierarchy have a short arrow, non-terminal nodes a long arrow. Those non-terminal nodes are expanded in following diagrams.

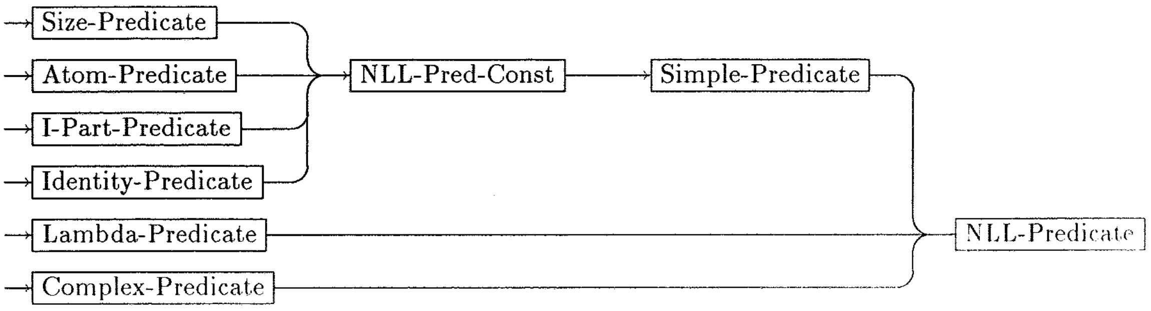
4.2.1 NLL-Domain

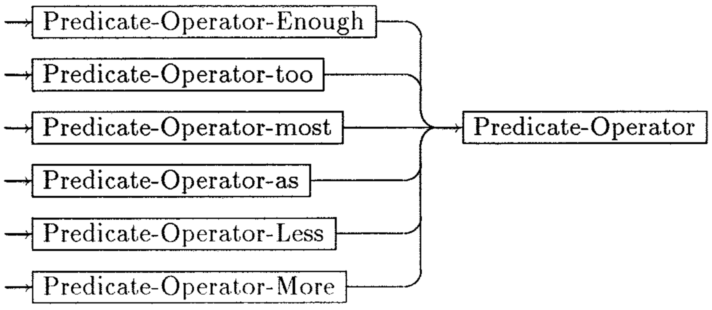


4.2.2 NLL-Expr

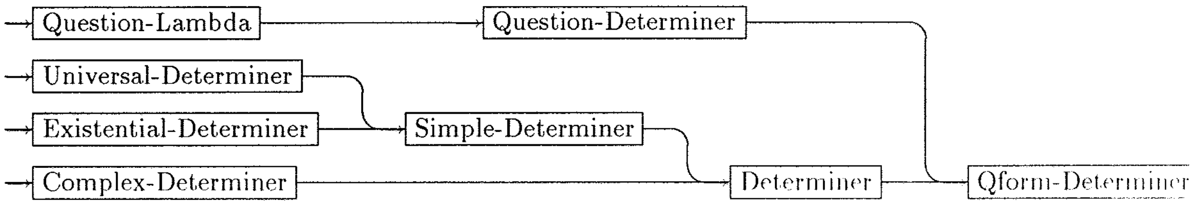

NLL-Term
NLL-Predicate
Predicate-Operator
Qform-Determiner
Conditional
NLL-Logical-Const
N-ary-Xor
N-ary-Iff
N-ary-Or
N-ary-Conn
N-ary-And
Questioner
Quantifier
Ternary-Quantifier
N-ary-Quantifier
Qform
Distributive-Operator
NLL-Expr
Qform-Wff
Atomic-Wff
Propositional-Variable
Atomic-False
NLL-Literal
NLL-Wff
Atomic-True
N-ary-Conn-Wff
Negated-Wff
Non-Atomic-Wff
Conditional-Wff
Generalized-Count
Generalized-Maximum
Generalized-Minimum
VB-TF-Operation
Generalized-Average
Generalized-Product
Generalized-Sum

4.2.3 NLL-Predicate and Predicate-Operator

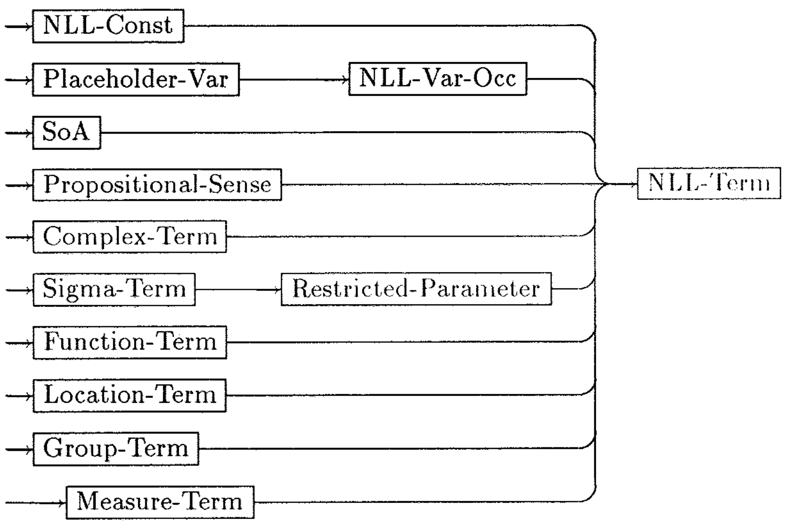


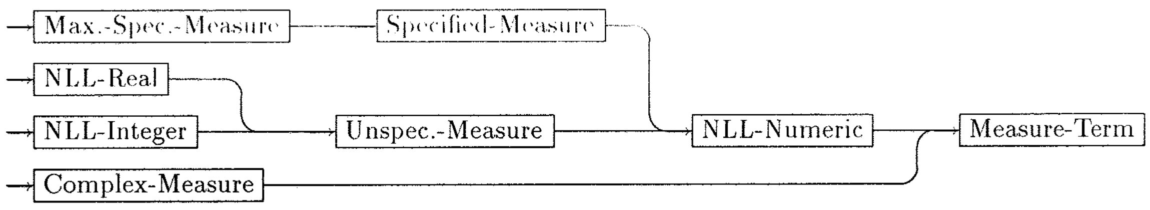


4.3 Qform-Determiner

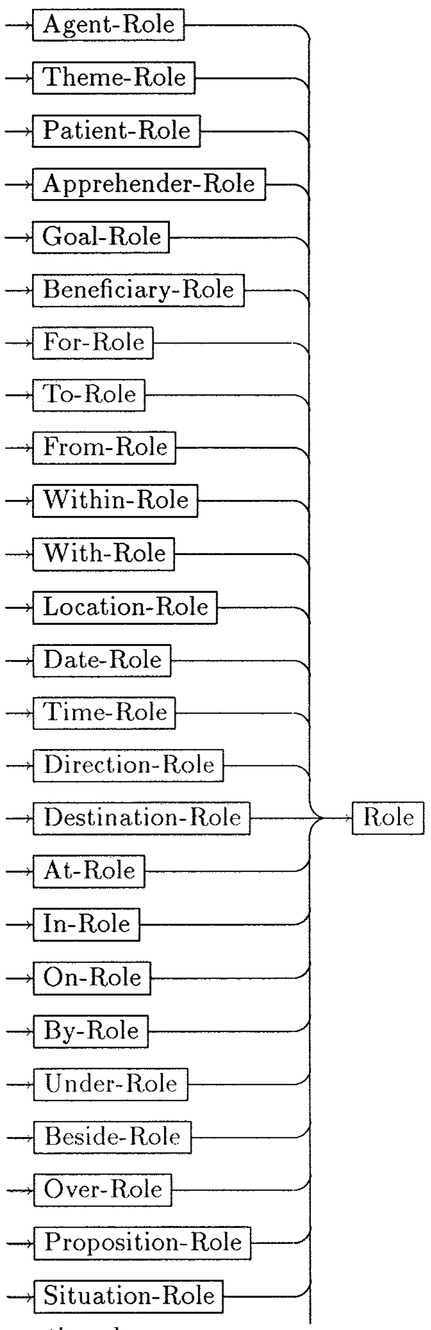


4.4 NLL-Term

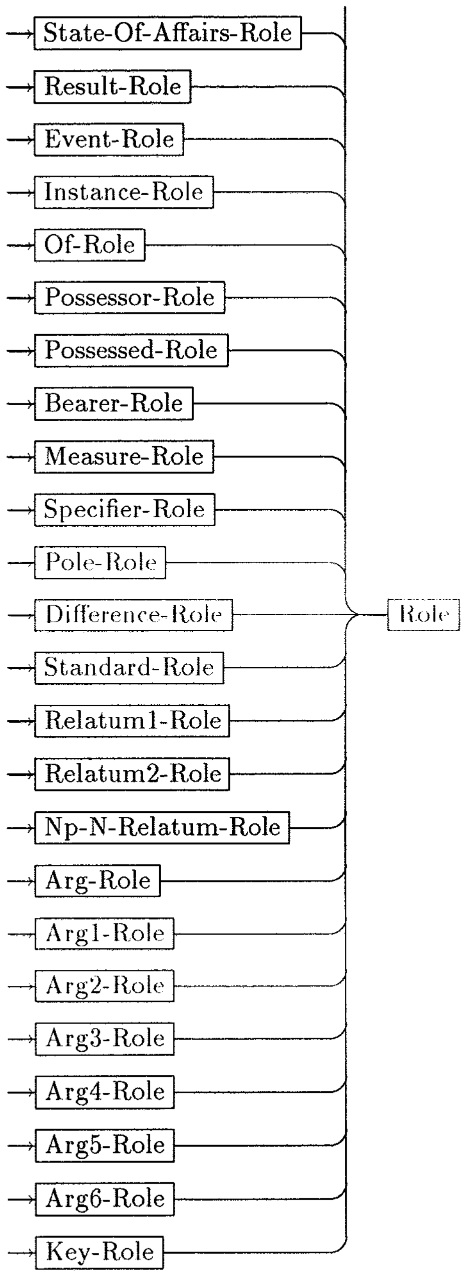

NLL-Const
Placeholder-Var
NLL-Var-Occ
SoA
Propositional-Sense
NLL-Term
Complex-Term
Sigma-Term
Restricted-Parameter
Function-Term
Location-Term
Group-Term
Measure-Term


Max.-Spec.-Measure
Specified-Measure
NLL-Real
NLL-Integer
Unspec.-Measure
NLL-Numeric
Measure-Term
Complex-Measure

4.5 Role



continued ...


State-Of-Affairs-Role
Result-Role
Event-Role
Instance-Role
Of-Role
Possessor-Role
Possessed-Role
Bearer-Role
Measure-Role
Specifier-Role
Pole-Role
Difference-Role
Standard-Role
Relatum1-Role
Relatum2-Role
Np-N-Relatum-Role
Arg-Role
Arg1-Role
Arg2-Role
Arg3-Role
Arg4-Role
Arg5-Role
Arg6-Role
Key-Role
Role

References

[1] H. Alshawi et al. *Research Programme in Natural Language Processing.* Final Report, Alvey Project No. ALV/PRJ/IKBS/105, SRI Cambridge Research Centre, July 1989.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers Principles, Techniques, and Tools.* Addison-Wesley Publ. Co., Reading, MA, 1986.

[3] Jon Barwise. *Noun Phrases, Generalized Quantifiers and Anaphora.* CSLI Report Ng. CSLI-86-52, Stanford, 1986.

[4] Jon Barwise and Robin Cooper. *Generalized Quantifiers and Natural Language.* Linguistics and Philosophy, Vol.4, 1981.

[5] Lewis G. Creary, J. Marc Gawron, and John Nerbonne. *Reference to Locations.* Proceedings of the Assoc. for Comp. Ling., 42-50, 1989.

[6] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, 4 edition, 1986.

[7] M. R. Genesreth et al. *Knowledge Interchange Format.* Logic Group Report, Computer Science Department, Stanford University, Stanford, 1991.

[8] G. Link. *The logical analysis of plurals and mass terms* In: R. Baeuerle, Ch. Schwarze, A. v. Stechow (eds.) *Meaning, Use and Interpretation of Language*, de Gruyter, Berlin 1987.

[9] John Nerbonne. *Nominal Comparatives and Generalized Quantifiers.* GWAI-90 Workshop on Plurals and Quantification. Springer Verlag, 1990.

[10] Dag Westerstähl. *Quantifiers in Formal and Natural Languages.* Rep. No. CSLI-86-55, Stanford University, Stanford, 1986. (Also in: D. Gabbay & F. Günthner (eds.) *Handbook of Philosophical Logic*, Vol. IV.)