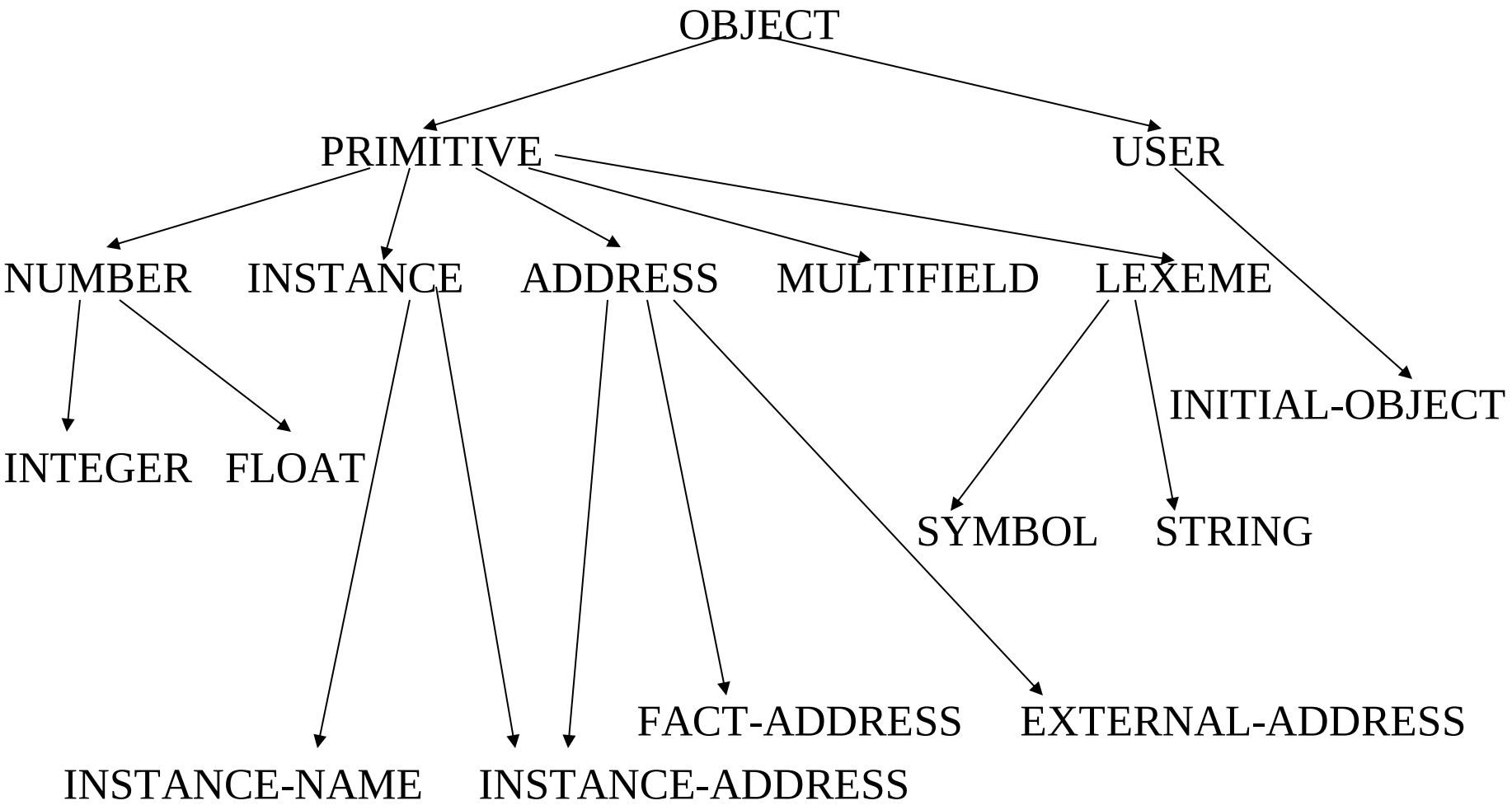
ОБЕКТНО ОРИЕНТИРАН ЕЗИК НА CLIPS (COOL)

Обектно ориентираният език на CLIPS (COOL) съчетава възможности и особености на различни обектно ориентирани системи и някои нови идеи.

ВГРАДЕНИ СИСТЕМНИ КЛАСОВЕ

COOL поддържа 17 системни класа, които не могат да бъдат унищожавани или модифицирани.

ЙЕРАРХИЯ МЕЖДУ ВГРАДЕНИТЕ КЛАСОВЕ



Дефиниране на класове

```
(defclass <name> [<comment>]
   (is-a <superclass-name>+)
   [<role>]
   [<pattern-match-role>]
   <slot>*
   <handler-documentation>*)

<role> ::= (role concrete|abstract)
<pattern-match-role> ::=
  (pattern-match reactive|non-reactive)
```

СЛОТОВЕ

```
<slot> ::=
  (slot <name> <facet>*) |
   (single-slot <name> <facet>*) |
   (multislot <name> <facet>*)
```

Името на даден слот може да бъде произволен символ с изключение на ключовите думи **is-a** и **name**, които са резервирани за използване в обектните образци.

Обектни образци в левите страни на правилата

```
<object-pattern> ::=
  (object <attribute-constraint>*)
<attribute-constraint> ::=
    (is-a <constraint>) |
    (name <constraint>) |
    (<slot-name> <constraint>*)
```

Тип на стойността на слота

Пример

```
> (clear)
> (defclass A (is-a USER)
      (role concrete)
      (multislot fff
         (create-accessor read)
         (default abc def ghi)))
> (make-instance a of A)
[a]
> (nth$ 2 (send [a] get-fff))
def
```

Фасети

```
<facet> ::=
  <default-facet> | <storage-facet> |
   <access-facet> | <propagation-facet> |
  <source-facet> |
  <pattern-match facet> |
  <visibility-facet> |
  <create-accessor facet> |
  <override-message facet> |
  <constraint-facets>
```

Фасета Default Value

Синтаксис:

```
(default ?DERIVE | ?NONE |
    <expression>*) |
(default-dynamic <expression>*)
```

Използва се за специфициране на началната стойност (стойността по подразбиране), която се дава на слота при създаване или инициализация на екземпляр на класа.

Фасета Storage

Синтаксис:

(storage local | shared)

Стойност на фасетата **local**: стойността на слота се съхранява с (в) екземпляра.

Стойност на фасетата **shared**: стойността на слота се съхранява с (в) класа.

Фасета Access

Синтаксис:

(access read-write | read-only | initialize-only)

Стойност **read-write**: означава, че в и от слота може да се пише и чете.

Стойност **read-only**: означава, че от слота може само да се чете.

Стойност **initialize-only**: означава, че слотът по принцип е read-only, с изключение на моментите на създаване или инициализиране на екземпляр.

Фасета Inheritance Propagation

Синтаксис:

(propagation inherit | no-inherit)

Стойност **inherit**: означава, че съответният слот в описвания клас може да бъде наследяван от екземпляри на други класове, които наследяват от този клас.

Стойност **no-inherit**: означава, че само директните екземпляри на описвания клас ще получават съответния слот.

Facета Source

Синтаксис:

(source exclusive | composite)

Стойност **exclusive**: при получаването на слотове от класовете-предшественици по време на създаването на екземпляри се вземат фасетите на най-специфичния клас, който е носител на слота, и се присвояват стойности по подразбиране на всички неспецифицирани фасети.

Стойност **composite**: фасетите, които не са специфицирани явно за най-специфичния клас, който дефинира слота, се вземат от следващия по специфичност родителски клас.

Фасета Pattern Match Reactivity

Синтаксис:

`(pattern-match reactive | non-reactive)`

Стойност **`reactive`**: слотът участва в съпоставянето и всяка промяна на стойността му (за даден екземпляр на определен реактивен клас) може отново да предизвика съпоставяне.

Стойност **`non-reactive`**: промените в стойността на слота не предизвикват ново съпоставяне.

Фасета Visibility

Синтаксис:

(visibility private | public)

Стойност **private**: определя, че само манипулаторите на съобщения на текущо дефинирания клас имат право на директен достъп до слота.

Стойност **public**: определя, че право на директен достъп до слота имат също и манипулаторите на съобщения на суперкласовете и подкласовете, които наследяват този слот.

Фасета Create-Accessor

Синтаксис:

```
(create-accessor ?NONE | read | write |
  read-write)
```

Предизвиква автоматично създаване на експлицитни манипулатори на съобщения за четене и/или писане на/в дадения слот. По подразбиране не се създават никакви манипулатори за достъп.

При стойност **read**: създава се манипулатор

```
(defmessage-handler <class>
   get-<slot-name> primary ()
     ?self:<slot-name>)
```

При стойност **write**: създава се манипулатор

```
(defmessage-handler <class>
   put-<slot-name> primary (?value)
       (bind ?self:<slot-name> ?value))
```

При стойност **read-write**: създават се и двата посочени манипулатора.

Фасета Override-Message

Синтаксис:

(override-message ?DEFAULT |
<message-name>)

По подразбиране функциите на COOL, които присвояват стойности на слотове чрез използване на съобщения (**make-instance** и др.), се опитват да присвоят стойност на слота с помощта на съобщението, наречено **put-<slot-name>**. Ако потребителят е избрал да използва не стандартните манипулатори за достъп, а други функции (манипулатори за достъп), трябва да използва разглежданата фасета. Тя определя какво съобщение ще се изпраща вместо стандартното.

Constraint фасети

Служат за задаване на ограничения върху стойността на слота, които могат да бъдат от няколко вида:

- допустим тип

(type <type-specification>)

- допустими стойности

(allowed-values <list>)

Служебната дума **values** може да бъде заменена с **numbers, integers, floats, symbols, strings** и т.н.

- допустим диапазон на числовите стойности

(range <range-specification>
<range-specification>)

Задава долната и горната граница на интервала, на който могат да принадлежат числовите стойности, записани в съответния слот.

<range-specification> е или число, или **?VARIABLE** (**?VARIABLE** означава -∞ като лява граница на дефиниционния интервал или +∞ като дясна граница на този интервал).

- допустим брой полета на multifield слот

(cardinality <cardinality-specification>
<cardinality-specification>)

Ограничава броя (от – до) на полетата, от които могат да бъдат съставени стойностите на даден multifield слот. Тук **<cardinality-specification>** е или число, или **?VARIABLE** (**?VARIABLE** означава 0 като минимален брой полета или +∞ като максимален брой полета). По премълчаване броят на полетата от стойността на един multifield слот е от 0 до +∞.

Документация на манипулаторите на съобщения

COOL позволява на потребителя да направи forward декларации на манипулаторите на съобщения за даден клас в рамките на конструкцията **`defclass`**. Тези декларации служат само за документация и се игнорират от CLIPS. За реално включване на манипулатори на съобщения към даден клас задължително трябва да бъдат използвани съответни **`defmessage-handler`** конструкции.

Пример

```
> (clear)
> (defclass rectangle (is-a USER)
   (role concrete)
   (slot side-a (default 1))
   (slot side-b (default 1))
   (message-handler find-area))
> (defmessage-handler rectangle
    find-area ()
     (* ?self:side-a ?self:side-b))
> (defmessage-handler rectangle
    print-area ()
     (printout t (send ?self find-area)
                 crlf))
```