Università degli Studi di Bari
Dipartimento di Informatica

Laboratorio di ICSE

CLIPS

- Parte 1 -

Nicola Di Mauro
ndm@di.uniba.it

Claudio Taranto
claudio.taranto@di.uniba.it

Contatti

- Contatti
  - Studio: stanza 508, V piano DIB
  - Telefono: 080 544 2297
  - email: claudio.taranto@di.uniba.it
  - Sito: http://www.di.uniba.it/~claudiotaranto/icse.html
  - email (oggetto: [ICSE] ...)

- Avvisi
  - Homepage
  - Studio

CLIPS: **C** **L**anguage **I**ntegrated **P**roduction **S**ystem

- CLIPS è un tool per lo sviluppo di Sistemi Basati su Conoscenza le cui origini risalgono al 1984 (NASA’s Johnson Space Center)

- Implementato in linguaggio C per eliminare i problemi legati all’uso del linguaggio LISP

CLIPS: **C** **L**anguage **I**ntegrated **P**roduction **S**ystem

- Linguaggio a regole basato sull'*algoritmo Rete* con il meccanismo di attivazione delle regole di tipo *Forward Chaining*

- La versione 5.0 (1991) introduce due nuovi paradigmi di programmazione
  - programmazione procedurale
  - programmazione orientata agli oggetti
    - CLIPS Object-Oriented Language (COOL)

Download CLIPS

http://clipsrules.sourceforge.net/

- CLIPS 6.3 Beta for Windows Release 3
- CLIPS 6.3 Beta for Mac OS X Release 1
- CLIPS Java Native Interface 0.3 Beta
- News and Information (2011-03-05)
- What is CLIPS?
- Download CLIPS
- Online Documentation
- Support Information
- CLIPS Expert System Group
- SourceForge Project Page
- Frequently Asked Questions
- Web Links

A Tool for Building Expert Systems

SOURCEFORGE.NET®

Last Update March 13, 2011 *Gary Riley*

Riferimenti:

CLIPS Reference Manual:

Volume I - The Basic Programming Guide
Volume II - The Advanced Programming Guide
Volume III – The Interfaces Guide

http://clipsrules.sourceforge.net/OnlineDocs.html

```
CLIPS is released as public domain software and as such
you are under no obligation to pay for its use. However,
if you derive commercial or monetary benefit from use of
the software or just want to show support, please
consider making a voluntary payment based on the worth of
the software to you as compensation for the time and
effort required to develop and maintain CLIPS. Payments
can be made online at http://order.kagi.com/?JKT.
```

CLIPS Online Documentation

Version 6.30 Beta

- User's Guide (HTML) (PDF)
- Basic Programming Guide (HTML) (PDF)
- Advanced Programming Guide (HTML) (PDF)
- Interfaces Guide (HTML) (PDF)

Version 6.24

- User's Guide (HTML) (PDF)
- Basic Programming Guide (HTML) (PDF)
- Advanced Programming Guide (HTML) (PDF)
- Interfaces Guide (HTML) (PDF)

Regole e funzioni in CLIPS

- CLIPS
  - un linguaggio per regole di produzione
  - un linguaggio procedurale

- Componente principale di un linguaggio rule-based
  - base dei fatti
    - rappresentano lo stato iniziale del problema
  - database delle regole
    - contengono gli operatori che trasformano lo stato del problema in una soluzione

- Motore inferenziale del CLIPS
  - match dei fatti con le regole
  - sceglie quali regole eseguire
  - esegue le azioni associate alla regola scelta

Jess

- Jess è un motore a regole basato su CLIPS per Java
  - http://www.jessrules.com/links/

Jess software, owned by Sandia National Laboratories, will be made available upon request at no cost to U.S. Federal Government Agencies for their own internal use. Sandia will also provide Jess upon request to Universities, Academic Institutions, and other U.S. National Laboratories, for their own internal research and development, through a no cost, restricted R&D license. Any other individual, internal or commercial use of Jess requires that you purchase a license.

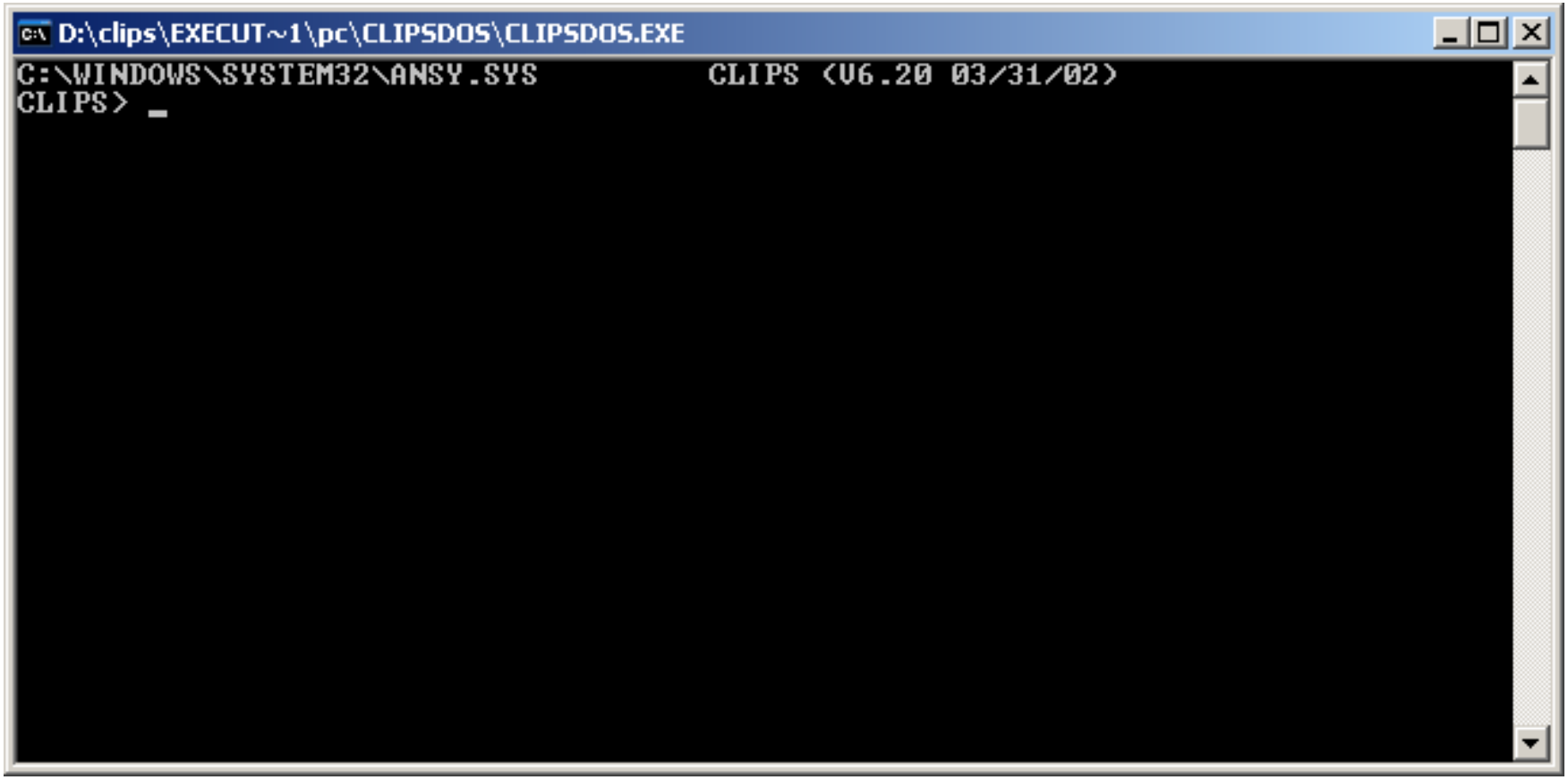
- Alcuni progetti basati su Jess
  - JadeJessProtege
    - Integrazione di JADE con JESS e Protege-2000
  - DAMLJessKB
    - reasoner per description logic, migrato in
  - OWLJessKB
    - A Semantic Web Reasoning Tool
  - SWRLJessTab
    - plug-in to the SWRLTab in Protege-OWL that supports the execution of SWRL rules using the Jess rule engine
- Alternativa
  - CLIPS Java Native Interface 0.3 Beta
    - http://clipsrules.sourceforge.net/CLIPSJNIBeta.html

Panoramica

Interagire con CLIPS

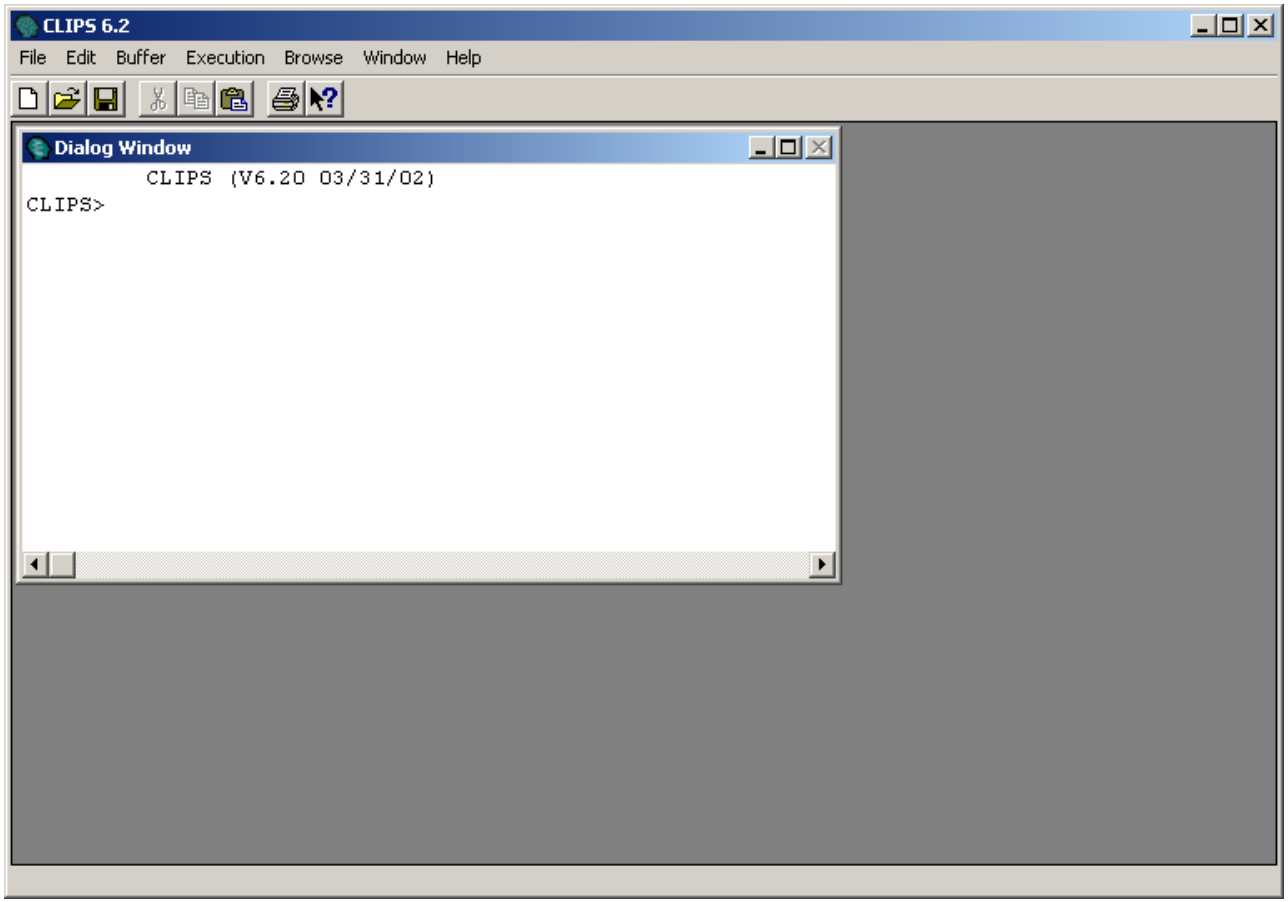
L’esecuzione di un programma CLIPS può avvenire

1. utilizzando una interfaccia testuale a linea di comando;



Iteragire con CLIPS

2. utilizzando una interfaccia a finestre;



3. come sistema esperto *embedded*.

Primi comandi

I comandi sono sempre racchiusi fra parentesi

- (load) carica un file .clp
- (clear) rimuove tutte le regole e i fatti dalla memoria
- (reset) rimuove solo i fatti dalla memoria
- (run) avvia l'esecuzione del programma
- ...

Basic Programming Elements

Il CLIPS fornisce tre elementi base per scrivere programmi

- tipi di dati primitivi
- funzioni per la manipolazione dei dati, e
- costrutti per l'aggiunta di conoscenza.

Tipi di dati per la rappresentazione di valori numerici

- integer: `237, 15, +12, -32`
- float: `237e3, 15.09, +12.0, -32.3e-7`

```
<integer> ::= [+ | -] <digit>+
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<float> ::= <integer> <exponent> |
       <integer> . [exponent]
       . <unsigned integer> [exponent]
       <integer> . <unsigned integer> [exponent]
<unsigned-integer> ::= <digit>+
<exponent> ::= e | E <integer>
```

Basic Programming Elements

Tipi di dati per la rappresentazione di valori simbolici

Symbol

- sequenza di caratteri
- non può iniziare con i caratteri ? e $?
  - sono riservati alle variabili
- `Esempi: foo, Hello, B76-HI, bad_value, 127A, 456-93-039, @+=-%, 2each`

String

- insieme di caratteri racchiusa tra "
- `Esempi: "foo", "a and b", "1 number", "a\"quote“`

In CLIPS un segnaposto con un valore è detto **field**

Multifield value

- sequenza di zero o più single field values racchiusi tra parentesi tonde
- Esempi: (a) (1 bar foo) () (X 3.0 “red” 567)

- Costante

- singolo field con valore fisso

- Variabile

- locazione simbolica utilizzata per immagazzinare valori

- Commenti

- preceduti dal simbolo ;

external-address

- indirizzi delle strutture dati esterne restituiti da una funzione integrata con il CLIPS

fact-adresses

- un fatto è una lista di valori atomici ai quali si può fare riferimento con:
  - posizione (*ordered facts*)
  - nome (*non-ordered* o *template facts*)

instance-name, instance-address

- una istanza è un oggetto di una specifica classe alla quale si può far riferimento con il suo nome

Oggetti CLIPS

- float, integers, symbol, string, multifield, external-adresses, fact-adresses o instances di una classe user-defined

Basic Programming Elements

instance-name

- simbolo racchiuso tra parentesi quadre

```
[pump-1] [foo] [+++] [123-890]
```

field

- In CLIPS un segnaposto con un valore è detto **field**

multifield value

- sequenza di zero o più single field values racchiusi tra parentesi tonde

```
(a) (1 bar foo) () (x 3.0 "red" 567)
```

Constant

- singolo field con valore fisso

Variable

- locazione simbolica utilizzata per immagazzinare valori

Commenti

- preceduti dal simbolo ;

Basic Programming Elements

funzione

- porzione di codice eseguibile identificata da uno specifico nome che restituisce un valore o ha un effetto collaterale

deffunction

- il costrutto **deffunction** permette all'utente di definire nuove funzioni, direttamente nell'ambiente CLIPS
  - funzioni interpretate dall'ambiente CLIPS

funzioni generiche

- le funzioni generiche possono essere definite utilizzando i costrutti **defgeneric** e **defmethod**
  - funzioni che permettono l'esecuzione di diverse porzioni di codice in base agli argomenti passati alla funzione
    - **overloading**

Clips utilizza la notazione prefissa per le funzioni

```
(* 8 (+ 3 (* 2 3 4) 9) (* 3 4))
```

Basic Programming Elements

Costrutti CLIPS

- **defmodule**,
- **defrule**,
- **deffacts**,
- **deftemplate**,
- **defglobal**,
- **deffunction**,
- **defclass**,
- **definstances**,
- **defmessage-handler**,
- **defgeneric,**
- **defmethod**.

Astrazione dati

La rappresentazione dell'informazione in CLIPS avviene per mezzo di fatti, oggetti e variabili globali.

Fatti

- I fatti sono una delle forme base di alto livello per la rappresentazione dell'informazione in CLIPS
- Ogni **fatto** rappresenta una porzione di informazione posta nella lista dei fatti, detta **fact-list**. I fatti sono l'unità fondamentale di dati utilizzata dalle regole.

I fatti possono essere aggiunti alla fact-list (utilizzando il comando **assert**), rimossi dalla fact-list (utilizzando il comando **retract**), modificati (utilizzando il comando **modify**), o duplicati (utilizzando il comando **duplicate**) con l'interazione esplicita dell'utente o da un programma CLIPS in esecuzione.

Per poter utilizzare i comandi **retract**, **modify**, e **duplicate** è necessario specificare un fatto per mezzo del suo **fact-index** o **fact-address**.

Ad ogni fatto aggiunto (o modificato) viene associato un indice intero (unico) detto fact-index

- i fact-index partono da zero e sono incrementati di uno per ogni nuovo fatto cambiato
- l'effetto dei comandi **reset** o **clear** è di porre a zero i fact-index.

Ordered Facts

Gli **Ordered facts** sono costituiti da un simbolo seguito da una sequenza di zero o più campi separati da spazi e delimitati da una parentesi aperta a sinistra e da una parentesi chiusa a destra.

Il primo campo di un ordered fact specifica una *relazione* che è applicata ai restanti campi del ordered fact.

Per esempio, `(father-of jack bill)` dice che `bill` è il padre `jack`.

```
(the pump is on)
(altitude is 10000 feet)
(grocery-list bread milk eggs)
```

Ordered Facts

CLIPS mantiene in memoria:

- una lista di fatti
- un insieme di regole che operano sui fatti

Per creare i fatti si utilizza il comando assert

CLIPS> (assert (colore blu))

<Fact-0>

Il fatto numero 0 è stato inserito nel database dei fatti, infatti

CLIPS>(facts)

f-0 (colore blu)

For a total of 1 fact

Ordered Facts

Asseriamo un altro fatto:

CLIPS>(assert (colore rosso))

<Fact-1>

E' possibile rimuovere un fatto dalla lista dei fatti utilizzando il comando retract <indice-fatto>

CLIPS>(retract 0)

CLIPS>(facts)

f-1 (colore rosso)

For a total of 1 fact.

I fact-index non vengono riusati.

Non-Ordered Facts

Gli Ordered facts rappresentano *informazione ordinata*. *Per poter accedere a questo tipo di informazione bisogna conoscere i campi che la contengono.*

I **Non-ordered (o deftemplate) facts** forniscono la possibilità di astrarre dalla struttura del fatto

- assegnando dei nomi ad ogni campo nel fatto

Il costrutto **deftemplate**

- analogo alla definizione di record o di struttura in C
- viene utilizzato per creare un template che poi può essere utilizzato per accedere ai campi per nome.

Un template è definito da zero o più definizioni di **named fields** o **slots**.

```
(deftemplate corso
    (slot nome)
    (slot studenti)
    (slot docente)
    (slot aula))
```

Istanziazione singola di un template

```
(assert (corso (nome ASS) (docente "Bianchi") (studenti 30) (aula "37A") ) )
(assert (corso (nome TIT) (studenti 35) (docente "Rossi") (aula "37A") ) )
```

Istanziazione di template mediante “deffacts”

```
(deffacts elenco_corsi
(corso (nome pippo) (studenti 15) (docente pluto) (aula 3b) )
(corso (nome ciccio) (studenti 18) (docente pluto) (aula 3c) )
)
```

Objects

Un **oggetto** in CLIPS è un simbolo, una stringa, un numero floating-point o intero, un multifield value, un external-address o una istanza di una classe user-defined.

Gli oggetti sono descritti da

- una proprietà e
- da un comportamento.

Una **classe** è un template di proprietà e comportamenti comuni ad oggetti che sono **istanze** della classe

```
Rolls-Royce della classe SYMBOL
"Rolls-Royce" della classe STRING
8.0 della classe FLOAT
8 della classe INTEGER
(8.0 Rolls-Royce 8 [Rolls-Royce]) della classe MULTIFIELD
<Pointer- 00CF61AB> della classe EXTERNAL-ADDRESS
[Rolls-Royce] della classe CAR (a user-defined class)
```

Global Variables

Il costrutto **defglobal** permette di definire variabili globali

- è possibile accedere dovunque a queste variabili
- alcuni costrutti (come defrule e deffunction) permettono l'allocazione di variabili locali alle quali si può accedere solo dall'interno del costrutto

Una variabile globale in CLIPS

- è simile alle variabili globali dei linguaggi procedurali
- ha tipizzazione debole
  - non necessariamente deve contenere un valore di un solo tipo

Rappresentazione della Conoscenza

Per rappresentare la conoscenza CLIPS fornisce

- un paradigma euristico e
- un paradigma procedurale.

Conoscenza Euristica: le regole

Un metodo per rappresentare la conoscenza in CLIPS è attraverso la regola.

Le regole sono utilizzate per rappresentare euristiche che specificano un insieme di azioni da effettuare in una data situazione.

Una regola è costituita da un antecedente (left-hand-side, LHS) e da un conseguente (right-hand-side, RHS)

Antecendente di una regola

L’antecedente di una regola è un insieme di **condizioni** (o **conditional elements**) che devono essere soddisfatte affinché la regola sia applicabile

- in CLIPS, la soddisfacibilità delle condizioni di una regola è basata sull’esistenza o meno di specifici fatti nella fact-list o di specifiche istanze di classi user-defined nella instance-list

Un tipo di condizione che può essere specificata è il **pattern**

- i pattern sono costituiti da un insieme di restrizioni, utilizzate per determinare quali fatti o oggetti soddisfano le condizioni nel pattern

Il processo di matching fra fatti/oggetti e il pattern è detto **pattern-matching**

- il CLIPS fornisce un meccanismo, **inference engine**, che automaticamente *matches* pattern sulla fact-list e sulla instance-list corrente, determinando quali regole sono applicabili

Conseguente di una regola

Il conseguente di una regola è costituito da un insieme di azioni da eseguire quando il motore inferenziale del CLIPS decide eseguire le regole applicabili

- se sono applicabili più regole, il motore inferenziale utilizza una **conflict resolution strategy** per selezionare quale regola deve essere eseguita
- vengono eseguite le azioni della regola selezionata (che possono influenzare la lista delle regole applicabili) e il motore inferenziale seleziona un'altra regola da eseguire
- questo processo continua fin quando non restano regole applicabil

Conoscenza Procedurale

Il CLIPS supporta anche un paradigma procedurale per la rappresentazione della conoscenza come quello dei linguaggi Pascal e C.

Le deffunctions e le funzioni generiche

- permettono all'utente di definire nuovi elementi eseguibili che provocano un side-effect oppure che restituiscono un valore

I message-handlers

- permettono all'utente di definire il comportamento degli oggetti specificando la loro risposta ai messaggi

Deffunctions, generic functions e message-handlers

- sono porzioni di codice procedurale, specificate dall'utente, che il CLIPS esegue quando opportuno

Defmodules

- permette di partizionare la base di conoscenza

Il costrutto deftemplate

Deftemplate

Ordered fact

- rappresentano informazione posizionale
- Per accedere a questo tipo di informazione, l'utente deve conoscere non solo il tipo di dato memorizzato ma anche quali campi lo contengono

Non-ordered (o deftemplate) fact

- forniscono la possibilità di astrarre dalla struttura dei fatti assegnando dei nomi ad ogni campo del fatto

Il costrutto **deftemplate**

- viene utilizzato per creare un template che può essere usato nei non-ordered fact per accedere ai campi per nome

Deftemplate

```
(deftemplate <deftemplate-name> [<comment>]
       <slot-definition>*)
<slot-definition> ::=
       <single-slot-definition> | <multislot-definition>
<single-slot-definition> ::=
       (slot <slot-name> <template-attribute>*)
<multislot-definition> ::=
       (multislot <slot-name> <template-attribute>*)
<template-attribute> ::=
       <default-attribute> <constraint-attribute>
<default-attribute> ::=
       (default ?DERIVE | ?NONE | <expression>*) |
       (default-dynamic <expression>*)


(deftemplate object
       (slot name)
       (slot location)
       (slot on-top-of)
       (slot weight)
       (multislot contents))
```

Slot Default Values

<default-attribute>

- specifica il valore da utilizzare come default quando viene asserito un template

L'attributo **default** specifica un valore di default statico

- la valutazione avviene una sola volta alla definizione del deftemplate

L'attributo **default-dynamic** è dinamico

- la valutazione avviene ogni volta che viene asserito un template fact

Slot Default Values: Esempio

```
CLIPS> (clear)
CLIPS>
(deftemplate foo
       (slot w (default ?NONE))
       (slot x (default ?DERIVE))
       (slot y (default (gensym*)))
       (slot z (default-dynamic (gensym*))))
CLIPS> (assert (foo))
[TMPLTRHS1] Slot w requires a value because of its
(default ?NONE) attribute.
CLIPS> (assert (foo (w 3)))
<Fact-0>
CLIPS> (assert (foo (w 4)))
<Fact-1>
CLIPS> (facts)
f-0 (foo (w 3) (x nil) (y gen1) (z gen2))
f-1 (foo (w 4) (x nil) (y gen1) (z gen3))
For a total of 2 facts.
CLIPS>
```

Regole per determinare i valori di default

1) Il tipo di default per lo slot è scelto dalla lista dei tipi:SYMBOL, STRING, INTEGER, FLOAT, INSTANCE-NAME , INSTANCE -ADDRESS , FACT-ADDRESS, EXTERNAL-ADDRESS.

2) Se il tipo di default ha una allowed constant restriction, allora viene scelto come default il primo valore specificato nello allowed constant attribute

3) Se il valore di default non è stato specificato al passo 2 e il tipo di default è INTEGER o FLOAT ed è specificato il range attribute, allora viene utilizzato come valore di default il minimo valore dell'intervallo se non è ? VARIABLE, altrimenti il massimo se non è ?VARIABLE.

Regole per determinare i valori di default

4) Se il valore di default non è stato specificato né al passo 2 né al passo 3, allora viene utilizzato il valore di default default. Esso assume valore nil per il tipo SYMBOL, "" per il tipo STRING, 0 per il tipo INTEGER, 0.0 per il tipo FLOAT, [nil] per il tipo INSTANCE-NAME, un puntatore ad una istanza fittizia per il tipo INSTANCE-ADDRESS, un puntatore ad un fatto fittizio per il tipo FACT-ADDRESS, e il puntatore NULL per il tipo EXTERNAL-ADDRESS.

Regole per determinare i valori di default

5) Se il valore di default da derivare è per un single field slot, allora viene usato il valore derivato dai passi 1 a 4. Il valore di default per un multifield slot è un multifield value di lunghezza zero. Se il multifield slot ha una cardinalità minima maggiore di zero, allora viene creato un multifield value con una lunghezza della minima cardinalità.

Slot Value Constraint Attribute

```
(deftemplate object
       (slot name
              (type SYMBOL)
              (default ?DERIVE))
       (slot location
              (type SYMBOL)
              (default ?DERIVE))
       (slot on-top-of
              (type SYMBOL)
              (default floor))
       (slot weight
              (allowed-values light heavy)
              (default light))
       (multislot contents
              (type SYMBOL)
              (default ?DERIVE)))
```

Il costrutto deffacts

Deffacts

Il costrutto **deffacts**

- permette la definizione di un lista di fatti che vengono asseriti automaticamente quando si invoca il comando **reset**
- i fatti asseriti con deffacts possono essere ritrattati o *pattern-matched* come gli atri fatti
- la initial fact-list, che include ogni deffacts definito, viene sempre ricostruita dopo ogni comando **reset**

Sintassi

```
(deffacts <deffacts-name> [<comment>] <RHS-pattern>*)
```

Deffacts

```
(deffacts startup "Refrigerator Status“
      (refrigerator light on)
      (refrigerator door open)
      (refrigerator temp (get-temp)))
```

Dopo ogni avvio e dopo il comando **clear**, CLIPS costruisce automaticamente il seguente deftemplate e deffacts.

```
(deftemplate initial-fact)
(deffacts initial-fact
      (initial-fact))
```

Il costrutto defrule

Defrule

Uno dei principali metodi per rappresentare conoscenza in CLIPS è attraverso la regola.

Una **regola** è una collezione di

- condizioni ed
- azioni da compiere se le condizioni sono soddisfatte

Lo sviluppatore di un sistema esperto definisce le regole che descrivono come risolvere un problema.

L'esecuzione delle regole (**fire**) è basata sull'esistenza o meno di fatti o istanze delle classi definite dall'utente

- CLIPS fornisce il meccanismo (**inference engine**) che cerca di eseguire un matching fra le regole e lo stato attuale del sistema (come rappresentato dalla fact-list e dalla instance-list) ed applica le azioni

Utilizzeremo il termine **pattern entity** per far riferimento o ad un fatto o ad una istanza di una classe definita dall'utente

Defining Rules

Le regole sono definite utilizzando il costrutto **defrule**

```
(defrule <rule-name> [<comment>]
      [<declaration>]              ; Rule Properties
      <conditional-element>*       ; Left-Hand Side (LHS)
      =>
      <action>*)                   ; Right-Hand Side (RHS)
```

Le azioni vengono eseguite in modalità sequenziale, se e solo se, sono soddisfatte tutte le condizioni (`conditional-element`) in LHS

```
(defrule example-rule
      "This is an example of a simple rule"
       (refrigerator light on)
       (refrigerator door open)
      =>
       (assert (refrigerator food spoiled)))
```

Basic cycle of Rule Execution

Costruita la base di conoscenza (in forma di regole) e preparata la fact-list e/o la instance-list, il CLIPS è pronto per eseguire le regole.

Nei linguaggi di programmazione convenzionali, il programmatore definisce esplicitamente la sequenza delle operazioni.

Ciò non accade in CLIPS.

Conoscenza (regole) e dati (fatti e istanze) sono separati, e il motore inferenziale fornito dal CLIPS viene utilizzato per applicare la conoscenza ai dati

Basic cycle of Rule Execution

a) Si considera la lista delle regole poste nell'agenda appartenenti al modulo corrente definito nel focus stack
   - Si seleziona quella che si trova al top della lista e la si esegue
     - Se non ci sono regole da attivare:
       - Si elimina il modulo corrente dal focus stack
       - Si passa il controllo al modulo successivo:
         - Se il focus stack è vuoto l'esecuzione termina, altrimenti si attivano le regole del nuovo modulo corrente

b) Si esegue la RHS della regola selezionata per l'esecuzione

Basic cycle of Rule Execution

c) il risultato del passo b) comporta la modifica della memoria di lavoro
  - questo provoca l'attivazione di altre regole o la disattivazione di quelle già presenti nell'agenda
  - le regole attivate vengono messe nell'agenda del modulo in cui sono definite e l'ordine è determinato:
    - Salience, e
    - Conflict resolution stategy
  - le regole disattivate vengono eliminate dall'agenda

d) se è stata definita una rivalutazione dinamica della salience, allora i valori delle salience appartenenti alle regole esistenti nell'agenda, vengono rivalutati; si riprende dal punto a)

Conflict Resolution Strategy

agenda

- lista di tutte le regole che hanno la parte sinistra (LHS) soddisfatta (e non sono ancora state eseguite)
- ogni modulo ha la sua propria agenda
- agisce come uno stack
  - la regola al top è la prima ad essere eseguita

Quando una regola viene attivata, la sua posizione nell'agenda è basata sui seguenti fattori:

- al di sopra di tutte le regole con *salience* più basso ed al di sotto di tutte le regole con *salience* più alto;
- fra regole con stesso *salience*, è la strategia di risoluzione del conflitto (conflict resolution strategy) a determinare la posizione;
- se una regola è attivata (insieme ad altre) per la stessa modifica della WM, e i passi a) e b) non sono in grado di specificare l'ordine, allora viene ordinata in base alla sistemazione delle altre regole attivate insieme ad essa

Salience

L’utente può assegnare una priorità ad una regola tramite la **salience** rule property.

I valori di salience possono essere valutati: quando viene definita la regola, quando viene attivata la regola o ad ogni esecuzione.

Valutazione Saliance alla definizione della regola

Valutazione Saliance alla attivazione della regola

```
(defrule test-1
      (declare (salience 99))
      (fire test-1)
      =>
      (printout t "Rule test-1 firing." crlf))
(defrule test-2
      (declare (salience (+ ?*constraint-salience* 10)))
      (fire test-2)
      =>
      (printout t "Rule test-2 firing." crlf))
```

Conflict Resolution: Depth Strategy

Le regole attivate sono posizionate sopra tutte le regole con lo stesso salience.

il `fatto-a` attiva la `regola-1` e la `regola-2`

il `fatto-b` attiva la `regola-3` e la `regola-4`

Allora se viene asserito il `fatto-a` prima del `fatto-b`

- la `regola-3` e la `regola-4` saranno poste al di sopra della `regola-1` e della `regola-2` nell'agenda
- Però la posizione, della `regola-1` relativa alla `regola-2` e la `regola-3` relativa alla `regola-4` sarà arbitraria

Conflict Resolution: Breadth Strategy

Le regole attivate sono posizionate al di sotto di tutte le regole con lo stesso salience.

il `fatto-a` attiva la `regola-1` e la `regola-2`

il `fatto-b` attiva la `regola-3` e la `regola-4`

Allora se viene asserito il `fatto-a` prima del `fatto-b`

- la `regola-1` e la `regola-2` saranno poste al di sopra della `regola-3` e della `regola-4` nell'agenda.
- Però la posizione, della `regola-1` relativa alla `regola-2` e la `regola-3` relativa alla `regola-4` sarà arbitraria.

Conflict Resolution: Simplicity - Complexity

Simplicity Strategy

Fra tutte le regole con lo stesso salience, le nuove regole attivate sono posizionate al di sopra di tutte le attivazioni di regole con maggiore o uguale *specificità*.

- La **specificity** di un regola è determinata dal numero di confronti da effettuare nella LHS della regola.

Complexity Strategy

Fra tutte le regole con stesso salience, le nuove regole attivate sono poste al di sopra di tutte le attivazioni di regole con minore o uguale specificità.

Conflict Resolution: LEX Strategy

Fra tutte le regole con lo stesso salience, le nuove regole attivate sono poste utilizzando la strategia OPS5.

- Ogni fatto ed ogni istanza sono etichettati con un “time tag” per indicare quanto siano *recenti* rispetto agli altri fatti e istanze nel sistema.
- I pattern entities associati ad ogni attivazione di regola sono ordinati in modo decrescente per determinare la posizione tra le regole con lo stesso salience.
- Un’attivazione con un pattern entities più recente è sistemata prima di una attivazione con pattern entities meno recenti.

Conflict Resolution: MEA Strategy

Fra tutte le regole con lo stesso salience, le nuove regole attivate sono posizionate utilizzando la strategia OPS5.

- Il primo “time tag” del pattern entity associato con il primo pattern è utilizzato per determinare dove inserire l’attivazione.

- Un’attivazione, il cui primo “time tag” del pattern è più grande di altri primi tag delle attivazioni, è inserita prima delle altre attivazione nell’agenda.

Conflict Resolution: Random Strategy

Ad ogni attivazione viene assegnato un numero casuale utilizzato per determinare la sua posizione dell'attivazione fra tutte quelle con lo stesso salience.

Questo numero casuale viene ricordato quando viene cambiata la strategia in modo che quando viene selezionata di nuovo la strategia random viene riprodotto lo stesso ordinamento.