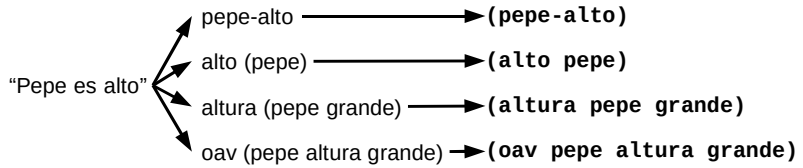
Representaciones basadas en reglas

- Base de conocimiento constituido fundamentalmente por
  - Hechos
    Afirmaciones incondicionales sobre algún aspecto del dominio del problema
    - Ej.: llueve
    - Ej.: recuento es inferior a 6000
  - Reglas
    Afirmaciones condicionales de la forma "**SI ... ENTONCES ...**"
    - Ej.: SI llueve, ENTONCES la calle está mojada
    - Ej.: SI presión mayor a 2000, ENTONCES activar alarma
  - Metarreglas
    Estructuras adicionales para controlar explícitamente el funcionamiento del motor de inferencia

Hechos

- Representa enunciados básicos del lenguaje natural



- Memoria trabajo tiene la lista de hechos
  - Iniciales
  - Obtenidos a partir de aplicar las reglas debido al funcionamiento del motor de inferencia

Reglas

- **Reglas declarativas**

  ("SI *p* es verdad, ENTONCES *q* también es verdad")

  - **Deductiva** (lógica clásica)
    Ej.: SI el recuento es inferior a 6000, ENTONCES hay leucopemia
  - **Causal** (causa → consecuencia)
    Ej.: SI llueve, ENTONCES la calle está mojada
  - **Abductiva** (síntomas → causa)
    Ej.: SI la calle está mojada, ENTONCES llueve

- **Reglas procedimentales**

  ("SI *p* es verdad, ENTONCES realizar la acción *a*")

  - **Exterior**
    Ej.: SI presión mayor a 2000, ENTONCES activar alarma
  - **Programa**
    Ej.: SI la fase es diagnosticar y ya tenemos un diagnóstico, ENTONCES pasar a la fase de establecer tratamiento

Motores de inferencia

- Motores de inferencia
  - Hacia adelante

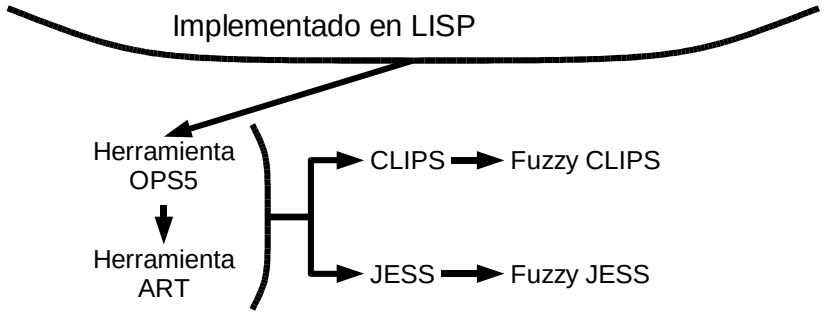
```
memoria-trabajo ← hechos-iniciales
mientras no configuración_final(memoria-trabajo)
   conjunto-reglas ← aplicable(memoria-trabajo, antecedentes)
   R ← resolver_conflictos(conjunto-reglas)
   C ← consecuente(R)
   memoria-trabajo ← mezclar(C, memoria-trabajo)
fin mientras
```

  - Hacia atrás

```
memoria-trabajo ← objetivos-iniciales
mientras no configuración_final(memoria-trabajo)
   objetivo ← seleccionar(memoria-trabajo)
   conjunto-reglas ← aplicable(objetivo, consecuentes)
   R ← resolver_conflictos(conjunto-reglas)
   A ← antecedente(R)
   memoria-trabajo ← mezclar(A, memoria-trabajo)
fin mientras
```

Lenguajes de reglas

- R1/XCON: sistema experto de configuración de los años 80

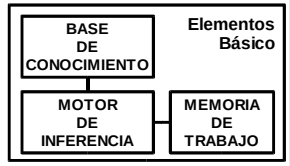


CLIPS (C Language Integrated Production System)

JESS (Java Expert System Shell)

CLIPS

- Entorno CLIPS
  - Base conocimiento: reglas y hechos inic.
  - Memoria de trabajo: lista de hechos
  - Motor de inferencia: encadenamiento hacia adelante




```
memoria-trabajo ← hechos-iniciales
mientras no configuración_final(memoria-trabajo)
   agenda ← activar(memoria-trabajo, antecedentes)
   R ← resolver_conflictos(agenda)
   C ← disparar(R)
   memoria-trabajo ← mezclar(C, memoria-trabajo)
fin mientras
```

  - Carga construcciones **`(def...)`**, órdenes y funciones
- Editor integrado
  - Carga construcciones (no funciones ni órdenes)
  - Balanceado de paréntesis

Definiciones básicas de un programa

- Tipos de hechos que van a manipularse
- Hechos iniciales del problema
- Reglas para manipular hechos (condición → acción)

- Notas:
  - Normalmente no se escriben directamente en el entorno
  - Suele usarse el editor. En el menú *File/Editor*
  - Extensión de los ficheros con programas CLIPS: ".clp"

Ejecución de un programa

1. Inicialización **`(reset)`**
   a) Borra contenido de la lista de hechos y la agenda
   b) Carga en la lista de hechos los hechos iniciales
   c) Carga en la agenda las reglas activadas
2. Evaluación (mientras la agenda no esté vacía) **`(run)`**
   a) Seleccionar la primera regla de la agenda
   b) Disparar la regla seleccionada
   c) Activar nuevas reglas

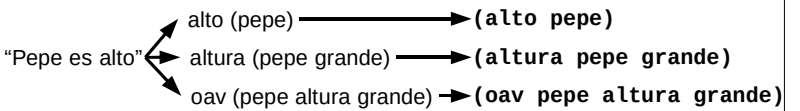
   - Nota:
     - la ordenación de las reglas en la agenda depende de la estrategia de control o estrategia de resolución de conflictos

Órdenes básicas ejecución

- Controlan la ejecución del programa
  - **(reset)**: inicialización
  - **(run)**: evaluación del programa hasta que se vacíe la agenda o encuentre **(halt)**
  - **(run x)**: x evaluaciones consecutivas
  - **(clear)**: limpia el entorno
- Notas:
  - Son órdenes y se cargan desde el entorno, no desde el editor
  - En el menú *Execution*
  - Para trazar el comportamiento de los programas es muy útil
    - Usar **(run 1)**
    - Visualizar ventanas de agenda y hechos. En el menú *Window*

Hechos (I)

- Memoria trabajo tiene la lista de hechos
- Tienen dirección e índice único
- Hecho ordenado
  - Sintaxis: **(<nombre-de-relación> <valor>*)**
    - Ej.: **(alarma-conectada)**
    - Ej.: **(temperatura 20)**
    - Ej.: **(es-un piolin animal)**
  - Representa enunciados básicos del lenguaje natural



Hechos (II)

- Hechos iniciales: cargar en la base de conocimiento
  (deffacts <nombre> [<comentario>] <hecho>*)
  - Ej.: **(deffacts hecho (alto pepe))**
  - Nota: los hechos iniciales se cargan después de ejecutar **(reset)**
    se carga un hecho inicial por defecto **(initial-fact)**
    <nombre> y el 1er **<hecho>** no pueden ser números
- **(facts)**: lista los hechos en la memoria de trabajo
- **(get-fact-list)**: lista las direcciones de los hechos
  - Ej.:

```
(facts)
(deffacts hecho0 (fase 2))
(deffacts hecho1 (nivel a))
(deffacts hecho2 (nivel c))
(reset)
(facts)
```

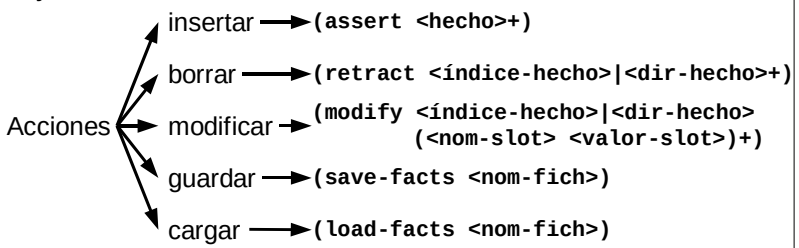

Hechos (III)

- Hechos iniciales: borrar de la base de conocimiento
  - Uno a uno
    (undeffacts <nombre>) o *Browse/Deffacts Manager*
  - Todos de golpe
    (clear) o *Execution/Clear*
  - Ej.:

```
(clear)
(deffacts hecho0 (fase 2))
(deffacts hecho1 (nivel a))
(reset)
(facts)
(undeffacts hecho2 (nivel c))
(reset)
(facts)
```

Hechos (IV)

- En ejecución



- Ej.:
```
(clear)
(reset)
(facts)
(assert (lcd 1))
(assert (disp 14))
(retract (1))
(facts)
```

- Ej.:
```
(save-facts "ff.txt")
(reset)
(facts)
(load-facts "ff.txt")
(facts)
```


Reglas (I)

- Reglas: en la base de conocimiento

```
(defrule <nom-regla> [<comentario>]
         <elemento-codicional>*
         =>
         <acción>*)
```

- Acciones con las reglas
  - Activar
    - Todos los elementos condicionales se satisfacen
    - La agenda recoge las reglas activas en cada momento
  - Disparar
    - Se ejecutan secuencialmente las acciones de la parte derecha de la regla

Reglas (II)

- **<acción>***: órdenes y funciones
- **<elemento-condicional>**:
  - Patrones:
    - Constantes
    - Variables
    - Restricciones
  - Ej.:
```
(clear)
(deffacts h0 (fase 2))
(defrule r0 (fase 2) => (assert (x0)))
(defrule reglaB => (reset))
(reset)
```


Elementos condicionales booleanos

- **(and <elemento-condicional>+)** = conjunción lógica
- **(or <elemento-condicional>+)** = disyunción lógica
- **(not <elemento-condicional>)** ≠ negación lógica
  - Lógica clásica:
    - Si no tenemos demostrado **p**, no podemos decir que ¬**p** esté demostrado
  - CLIPS:
    - No tener **p** en la memoria de trabajo ↔ se cumple **(not (p))**

Ejemplo-1

HECHOS EN LA MEMORIA DE TRABAJO:

```
(fase 2)(nivel a)   (nivel c)
(oav sensor1 presion alto)          (oav sensor1 estado dudoso)
(oav sensor2 temperatura normal)    (oav sensor2 estado ok)
```

PARTES IZQUIERDAS DE LAS REGLAS (LHR):

LHR Regla 0: `(fase 2)`

LHR Regla 1: `(oav sensor1 presion alto) (nivel b)`

LHR Regla 2: `(oav sensor1 presion alto) (or (fase 1) (nivel b))`

LHR Regla 3:
```
(or (and (oav sensor1 presion alto)
         (oav sensor1 estado ok))
    (and (oav sensor2 temperatura alto)
         (oav sensor2 estado ok)))
```

LHR Regla 4: `(not (oav sensor3 estado ok))`

LHR Regla 5: `(nivel a) (not (oav sensor3 estado ok)) (nivel c)`

Cargar ejemplo-1.clp y comprobar las ventanas de hechos y agenda

Restricciones conectoras

- **`<termino1> & <termino2> ... & <termino-n>`**

  se satisface si se satisfacen las dos restricciones adyacentes

- **`<termino1> | <termino2> ... | <termino-n>`**

  se satisface si se satisface alguna de las dos restricciones adyacentes

- **`~<termino1>`**

  se satisface si no se satisface la restricción adyacente

Ejemplo-2

HECHOS EN LA MEMORIA DE TRABAJO:

```
(fase 2)(nivel a)   (nivel c)
(oav sensor1 presion alto)          (oav sensor1 estado dudoso)
(oav sensor2 temperatura normal)    (oav sensor2 estado ok)
```

PARTES IZQUIERDAS DE LAS REGLAS (LHR):

LHR Regla 6: `(oav sensor1|sensor2 presion alto) (nivel ~b)`

LHR Regla 7: `(oav sensor3 estado ~ok)`

LHR Regla 8: `(nivel ~a)`

LHR Regla 9: `(not (nivel a))`

Cargar ejemplo-2.clp y comprobar las ventanas de hechos y agenda

Ejercicio-1

HECHOS EN LA MEMORIA DE TRABAJO:

```
(fase 1)   (nivel b)
(oav sensor1 presion normal)      (oav sensor1 estado ok)
(oav sensor2 presion alto)        (oav sensor2 estado dudoso)
(oav sensor3 temperatura alto)
```

PARTES IZQUIERDAS DE LAS REGLAS (LHR):

LHR Regla A: `(oav sensor1 presion ~alto) (nivel ~b)`

LHR Regla B: `(oav sensor2 ~presion ~alto)`

LHR Regla C: `(oav sensor1|sensor2 presion alto) (fase ~2)`

LHR Regla D:
```
(or (oav sensor4 presion alto|normal)
    (oav sensor3 estado ~ok)
    (and (oav sensor2 estado ~ok)
         (oav sensor2 presion|temperatura alto)))
```

Establecer la activación de las reglas teniendo en cuenta el contenido de la memoria de trabajo

Ejercicio-2 (I)

```
; Programa ejercicio-2.clp
; POSIBLES HECHOS INICIALES DEL PROGRAMA
; (estacion primavera|verano|otonyo|invierno): la estacion del año
; (actividad gastronomica|deportiva): tipo de actividad deseada
; POSIBLES HECHOS INFERIDOS POR EL PROGRAMA
; (destino campo|playa)  : posibles clases de destinos turisticos
; (zona   <nombre-de-zona>) : subclases de destinos turisticos
; (lugar   <nombre-de-lugar>) :instancias de destinos turisticos

;es primavera y deseamos hacer una excursion gastronomica
(deffacts hechos-iniciales
   (estacion primavera)
   (actividad gastronomica))

;si es primavera, los destinos posibles son campo y playa
(defrule posibilidades-primaverales
   (estacion primavera)
=>
   (assert (destino campo)
      (destino playa)))

;si un destino posible es playa, una zona recomendada es Malaga
(defrule playa-de-Malaga
   (destino playa)
=>
   (assert (lugar Malaga)))

;si un destino posible es playa, una zona recomendada es Torremolinos
(defrule playa-de-Torroles
   (destino playa)
=>
   (assert (lugar Torremolinos)))

;si un destino posible es campo y una actividad deseada es
      gastronomica,
;entonces una zona recomendada es iberico
(defrule campo-gastronomico
   (destino campo)
   (actividad gastronomica)
=>
   (assert (zona iberico)))

;si un destino posible es campo y una actividad deseada es deporte,
;entonces una zona recomendada es alta-montanya
(defrule campo-deportivo
   (destino campo)
   (actividad deporte)
=>
   (assert (zona alta-montanya)))

;si una zona posible es iberico, un lugar recomendado es Aracena
(defrule para-iberico-Aracena
   (zona iberico)
=>
   (assert (lugar Aracena)))

;si una zona posible es alta-montanya, un lugar recomendado es
      Aracena
(defrule Sierra-Nevada
   (zona alta-montanya)
=>
   (assert (lugar Sierra-Nevada)))
```


Ejercicio-2 (II)

- Modificar el programa para responder a las siguientes especificaciones:
  - Estamos en primavera y la actividad es deportiva
  - El programa debe pararse tras deducir el primer lugar recomendado
  - Deben expresarse también algún conocimiento acerca de los lugares recomendados en invierno

Ejercicio-3 (I)

Consideremos las siguientes reglas relativas a la calificación de los alumnos matriculados en *"Papiroflexia Avanzada"*:

Se supera la asignatura si se superan tanto la teórica como la práctica. Se supera la teórica si la calificación en cada uno de los dos exámenes teóricos ha sido "apto". Se supera la práctica si la calificación en cada una de las dos prácticas ha sido "apto", o bien si en una ha sido "no apto" y se ha realizado un trabajo complementario.

1. Dar un conjunto de objetos, atributos y valores adecuado para simbolizar toda esta información.
2. Representar la información anterior mediante un conjunto de reglas CLIPS, empleando un predicado "objeto-atributo-valor".
3. Escribir un programa CLIPS que calcule qué alumnos han superado la asignatura. Sabemos además que las calificaciones de un alumno de la clase han sido las siguientes:

T1	T2	P1	P2	TRABAJO
APTO	APTO	NO APTO	APTO	OK

   Introducir esta información en un deffacts y calcular mediante CLIPS si este alumno ha superado la asignatura.

4. Id. para otro alumno dado por

T1	T2	P1	P2
APTO	APTO	NO APTO	APTO

Ejercicio-3 (II)

Modificaciones al ejercicio 3 (I):
ahora se considera que las calificaciones posibles en cada examen y cada práctica son "suspenso", "aprobado", "notable" y "sobresaliente". Las reglas pasan a ser las siguientes:
Se supera la asignatura si se superan tanto la teórica como la práctica. Se supera la teórica si la calificación en cada uno de los dos exámenes teóricos ha sido distinta de "suspenso". Se supera la práctica si la calificación en cada una de las dos prácticas ha sido distinta de "suspenso", o bien si en una ha sido "suspenso" pero en otra ha sido "sobresaliente" o se ha realizado un trabajo complementario.

1. Suponiendo un solo alumno, dar un conjunto de atributos y valores adecuado para simbolizar toda esta información.
2. Representar la información anterior mediante un conjunto de reglas CLIPS, empleando un predicado "objeto-atributo-valor".
3. Escribir un programa CLIPS que calcule si el alumno ha superado la asignatura.
4. Sabemos además que las calificaciones del alumno han sido las siguientes:

T1	T2	P1	P2	TRABAJO
Apr.	Not.	Susp.	Not.	OK

Introducir esta información en un deffacts y calcular mediante CLIPS si este alumno ha superado la asignatura.